```
#include <senior week 9.5>
```

Knapsack

Knapsack is a special case of DP. The name comes from a particular problem that goes like this:

Given a backpack that can hold w weight and a set of items that each have a weight and a value, determine the highest value of items we can bring.

There are many variations of knapsack, for example: adding a second limiting factor (size, ex.), not caring about value, having infinite amounts of items, etc.

To solve these kinds of problems, we can use DP on the parameters given in the question. That is, if someone is going up stairs (like in the contest 🙂), that's what the size of the DP array is. DP[x] = # of ways to get to stair x. Again, this aligns with what DP is doing - solving a smaller subquestion.

Tip on quickly clearing an array:

```
memset(arr,0,sizeof(arr));
```

Do this: [CCC '00 S4 - Golf - DMOJ: Modern Online Judge]

Each club can be used an infinite amount of times.
Note, each club only has 1 thing - its weight. (Or you could say all value = 1)
So, we want to get the minimum value being used. Since value is the answer, our DP array should be something like arr[?] = Min value.

```
#include <bits/stdc++.h>

using namespace std;

int d,num,clubs[32],dis[5281];

int main() {
   cin >> d >> num;
   memset(dis,-1,sizeof(dis));
   dis[0] = 0;
   for (int i = 0; i < num; i ++){
       cin >> clubs[i];
   }
```

```
    for (int i = 1; i <= d; i ++){
        for (int j = 0; j < num; j ++){
            int club = clubs[j];
            if (i >= club && dis[i-club] != -1){
                if (dis[i] == -1) dis[i] = dis[i-club]+1;
                else dis[i] = min(dis[i],dis[i-club]+1);
            }
        }
    }

    if (dis[d] == -1){
        cout << "Roberta acknowledges defeat.";
    } else cout << "Roberta wins in " << dis[d] << " strokes.";

    return 0;
}
```

Try this: [Educational DP Contest AtCoder D - Knapsack 1 - DMOJ: Modern Online Judge]

This time, we have the same question except the items are given weights. Also, each item can only be used once.

Q1: How do we account for the fact that they are given different weights? Hint: what was the weight in the previous question? What did that affect?
Q2: How can we loop so that each item is always used at most once?

```
#include <bits/stdc++.h>

using namespace std;

//steps:
//input, memset
//choose item
//loop thru weights (w->0)
//transition
int n,w;
int items[100][2];
long long dp[100001];

int main() {
```

```
    cin >> n >> w;
    for (int i = 0; i < n; i ++){
        cin >> items[i][0] >> items[i][1]; //0 = weight, 1 = value
    }
    memset(dp,0,sizeof(dp));
    for (int i = 0, wi, vi; i < n; i ++){
        wi = items[i][0];
        vi = items[i][1];
        for (int j = w; j >= wi; j --){
            //j = rest+wi
            //dp[j] = dp[rest]+vi
            dp[j] = max(dp[j],dp[j-wi]+vi);
        }
    }

    cout << dp[w];
    return 0;
}
```

The same question? No. [Educational DP Contest AtCoder E - Knapsack 2 - DMOJ: Modern Online Judge]

What is the difference in this question?

DP Contest D:

Constraints

- All values in input are integers.
- $1 \leq N \leq 100$
- $1 \leq W \leq 10^5$
- $1 \leq w_i \leq W$
- $1 \leq v_i \leq 10^9$

DP Contest E:

Constraints

- All values in input are integers.
- $1 \leq N \leq 100$
- $1 \leq W \leq 10^9$
- $1 \leq w_i \leq W$
- $1 \leq v_i \leq 10^3$

The only difference is the constraints.
If we attempt the exact same approach as last time, what is the problem?
There are 3 things there: N (Number of items); for each item, W (weight) and V (value). Which of these do we want to DP (or recurse) on?
Last time it looked like this: DP[weight] = value. That's not an option now, as W can be as large as 10^9.

```
#include <bits/stdc++.h>

using namespace std;
```

```cpp
//steps:
//input, memset
//choose item
//loop thru values (n*max(v)->0). since n <= 100, v <= 1000 n*max(v) <= 1e5
//transition
int n,w;
int items[100][2];
long long dp[100001];

int main() {

    cin >> n >> w;
    for (int i = 0; i < n; i ++){
        cin >> items[i][0] >> items[i][1];
    }
    memset(dp,0x3f,sizeof(dp));
    dp[0] = 0;
    for (int i = 0, wi, vi; i < n; i ++){
        wi = items[i][0];
        vi = items[i][1];
        for (int j = 1e5; j >= vi; j --){
            //j = rest+vi
            //dp[j] = dp[rest]+wi
            dp[j] = min(dp[j],dp[j-vi]+wi);
        }
    }

    for (int j = 1e5; j >= 0; j--){
        if (dp[j] <= w){
            //first time we see a valid weight
            cout << j; return 0;
        }
    }
    return 0;
}
```

HW: [Coin Change - DMOJ: Modern Online Judge]
They forgot to put N <= 86.
[VM7WC '16 #5 Silver - Jayden Eats Chocolate - DMOJ: Modern Online Judge]
[Bank Notes - DMOJ: Modern Online Judge] (requires a lot of optimization, don't have to do it)
If you have trouble doing these, please ask for help in #senior-help or some other channel.
Ex. for Coin Change:

```cpp
#include <bits/stdc++.h>

using namespace std;

int x,n,coins[86],dp[10001];

int main() {
    memset(dp,-1,sizeof(dp));
    dp[0] = 0;
    cin >> x >> n;
    for (int i = 0; i < n; i ++){
        cin >> coins[i];
    }
    for (int j = 0,coin; j < n; j ++){
        coin = coins[j];
        for (int i = coin; i <= x; i ++){
            if (dp[i-coin] != -1){
                if (dp[i] == -1) dp[i] = dp[i-coin]+1;
                else dp[i] = min(dp[i],dp[i-coin]+1);
            }
        }
    }
    cout << dp[x];
    return 0;
}
```

[Coding Spree - DMOJ: Modern Online Judge]

Similar to Knapsack 1 - make sure you understand how to do that question!

```cpp
#include <bits/stdc++.h>

using namespace std;

#define pii pair<int,int>
int t,n, dp[1001];
pii problems[1000];

int main() {
   cin >> n >> t;
   memset(dp,0,sizeof(dp));

   for (int i = 0; i < n; i ++){
       cin >> problems[i].first >> problems[i].second;
   }
```

```
    for (int i = 0,v,h; i < n; i ++){
        v = problems[i].first; h = problems[i].second;
        for (int j = 1000; j >= h; j--){
            dp[j] = max(dp[j],dp[j-h]+v);
        }
    }

    cout << dp[t];

    return 0;
}
```

[DMOPC '18 Contest 4 P2 - Dr. Henri and Responsibility - DMOJ: Modern Online Judge]
The hardest part of the question is this: What is the target time?

```
#include <bits/stdc++.h>

using namespace std;

int n, tasks[700], tot = 0;
bool dp[245001];
//here, dp[v] = if I can achieve this time.
int main() {
    memset(dp,false,sizeof(dp));
    dp[0] = true;

    cin >> n;
    for (int i = 0; i < n; i ++) {
        cin >> tasks[i];
        tot += tasks[i];
    }
    for (int i = 0, task; i < n; i ++){
        task = tasks[i];
        for (int j = tot/2; j >= task; j--){
            if (dp[j-task]){
                dp[j] = true;
            }
        }
    }
    for (int i = tot/2; i >= 0; i --){
        if (dp[i]) {
            cout << tot-2*i;
            return 0;
```

```
        }
    }

}
```

[DMOPC '13 Contest 3 P5 - A Romantic Dinner - DMOJ: Modern Online Judge]
There are 3 attributes now: Value, Time, and Units (of food).
What are we being asked for? What are the constraints? DP doesn't have to be 1D !

```
#include <bits/stdc++.h>

using namespace std;

typedef long long ll;
typedef pair<int,int> pii;

int m,u,r; //minutes, units of food, restos
int dp[301][101];
int rests[150][3]; //Value Time Food
int32_t main(){

    cin >> m >> u >> r;
    for (int i = 0; i < r; i ++) cin >> rests[i][0] >> rests[i][1] >>
rests[i][2];

    memset(dp,0,sizeof(dp));
    for (int i = 0, tim,food,val; i < r; i++){
        tim = rests[i][1];
        food = rests[i][2];
        val = rests[i][0];
        for (int j = m; j >= tim; j --){
            for (int k = u; k >= food; k--){
                dp[j][k] = max(dp[j][k],dp[j-tim][k-food] + val);
            }
        }
    }
    cout << dp[m][u];

    return 0;
}
```