0 - 1 Knapsack Problem

> 0-1背包问题

二维数组空间

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

using namespace std;

typedef long long ll;

ll knapscak(vector<ll> &vals, vector<ll> &weigths, int N, int W) {
    vector<vector<ll>> dp(N + 1, vector<ll>(W + 1));
    for (int i = 1; i <= N; ++i) {
        for (int j = 1; j <= W; ++j) {
            j >= weigths[i - 1] ?dp[i][j] = max(dp[i - 1][j], dp[i - 1][j - weigths[i - 1]] + vals[i - 1]):
                    dp[i][j] = dp[i - 1][j];
        }
    }0
    return dp[N][W];
}

int main() {
    int T;
    scanf("%d", &T);
    while (T--) {
        int N, W;
        scanf("%d %d", &N, &W);
        vector<ll> vals(N), weights(N);
        for (int i = 0; i < N; ++i) scanf("%lld", &vals[i]);
        for (int i = 0; i < N; ++i) scanf("%lld", &weights[i]);
        printf("%lld\n", knapscak(vals, weights, N, W));
    }
}
```

优化

采用一维数组优化

由之前可知，dp[i][j]取决于dp[i - 1][j]和dp[i - 1][j -weight[i - 1]]

可以采用`滚动数组`

确保第i次循环结束后的dp[W]等于dp[i][W]即可

肯定是有一个主循环确保i从1~N循环的，然后对于背包体积，必须从V开始，逆序遍历，这样才能保证计算dp[v]时，dp[v - Ci]保存的是状态dp[i -1][v - Ci]的值

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

using namespace std;

typedef long long ll;

ll knapscak(vector<ll> &vals, vector<ll> &weigths, int N, int W) {
    vector<ll> dp(W + 1);
    for (int i = 1; i <= N; ++i) {
        for (int v = W; v >= weigths[i - 1]; --v) {
            dp[v] = max(dp[v], vals[i - 1] + dp[v - weigths[i - 1]]);
        }
    }
    return dp[W];
}

int main() {
    int T;
    scanf("%d", &T);
    while (T--) {
        int N, W;
        scanf("%d %d", &N, &W);
        vector<ll> vals(N), weights(N);
        for (int i = 0; i < N; ++i) scanf("%lld", &vals[i]);
        for (int i = 0; i < N; ++i) scanf("%lld", &weights[i]);
```

```
        printf("%lld\n", knapscak(vals, weights, N, W));
    }
}
```