Minimum cost to fill given weight in a bag

> 给定Wkg背包与物品cost数组，求装满Wkg的背包的最低物品cost （完全背包问题）

1. 首先排除掉不可用的物品cost，建立两个新数组availableCosts与availableWeight，availableCosts记录可用物品cost，availableWeight记录相应的重量
2. dp[i][j]表示前i个物品装进jkg的背包的最小花费
3. 从第一件物品开始遍历：
    1. 如果当前物品的重量大于大于当前背包容量，那么无法放入背包
    2. 如果当前物品重量小于背包容量，那么有两种情况：
        1. 不放入当前物品，那么 dp[i][j] = dp[i -1][j]
        2. 放入当前物品，dp[i][j] = dp[i][j - availableWeight[i - 1]] + dp[i][j - availableWeight[i - 1]] + availableCosts[i - 1] （完全背包问题）

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

using namespace std;
using vi = vector<int>;
using vvi = vector<vector<int>>;
#define INF 1000000
int minCosts(vi &costs, int N, int W) {
    vi availableCosts, availableWeight;
    for (int i = 0; i < N; ++i) {
        if (costs[i] != -1) {
            availableCosts.emplace_back(costs[i]);
            availableWeight.emplace_back(i + 1);
        }
    }
    int availableSize = availableCosts.size();
    vvi dp(availableSize + 1, vi(W + 1));
    for (int i = 0; i <= W; ++i) {
        dp[0][i] = INF;
    }
    for (int i = 1; i <= availableSize; ++i) {
        dp[i][0] = 0;
    }
    for (int i = 1; i <= availableSize; ++i) {
        for (int j = 1; j <= W; ++j) {
            availableWeight[i - 1] > j ? dp[i][j] = dp[i - 1][j] :
                    dp[i][j] = min(dp[i - 1][j], dp[i][j - availableWeight[i - 1]] + availableCosts[i - 1]);
        }
    }
    return (dp[availableSize][W] == INF) ? -1 : dp[availableSize][W];
}

int main() {
    int T;
    scanf("%d", &T);
    while (T--) {
        int N, W;
        scanf("%d %d", &N, &W);
        vi costs(N);
        for (int i = 0; i < N; ++i) scanf("%d", &costs[i]);
        printf("%d\n", minCosts(costs, N, W));
    }
    return 0;
}
```