Knapsack DP notes

Easy.

1. Dice combinations

$$dp_i = \sum_{j=1}^{6} dp_{i-j}$$

2. Knapsack 1:

define $dp_{i,j}$ as the maximum cost considering the first i elements with the remaining j weight.

$$dp_{i,j} = \max(dp_{i-1,j+w_i} + c_i,\ dp_{i-1,j})$$

base case: $dp_{0,W}$

ans: $\max\limits_{0 \le j \le W}(dp_{n,j})$

```cpp
void absher(){
    ll n, W;
    cin >> n >> W;
    for (ll i = 1; i <= n; i++) cin >> w[i] >> c[i];
    ll ans = 0;
    for (ll i = 0; i <= n; i++) for (ll j = 0; j <= W; j++) dp[i][j] = 0;
    for (ll i = 1; i <= n; i++){
        for (ll j = W; ~j; j--){
            if (j+w[i]<=W) dp[i][j] = max(dp[i][j],dp[i-1][j+w[i]]+c[i]);
            dp[i][j] = max(dp[i][j],dp[i-1][j]);
            if (i==n) ans=max(ans,dp[i][j]);
        }
    }
    cout << ans << "\n";
}
```

4. Frog 1

$$dp_i = \min(dp_{i-1} + |h_i - h_{i-1}|,\ dp_{i-2} + |h_i - h_{i-2}|)$$

* Minimizing Coins:

$dp_i =$ min coins needed to form sum i

$$dp_i = \min_{c \in C}(dp_{i-c} + 1)$$

* Removing digits

$dp_i \rightarrow$ min number of moves to reach number i

answer is dp_0

dp_i →
- dp_{i-d_1}
- dp_{i-d_2}
- dp_{i-d_3}
- ⋮

* Grid path 1

$dp_{i,j} \rightarrow$ number of ways to reach cell (i,j)

$dp_{i,j} = dp_{i-1,j} + dp_{i,j-1}$ / base case $dp_{0,0} = 1$

ans $= dp_{n-1,m-1}$

* Array description:

$dp_{i,j} \rightarrow$ number of arrays of size i putting integer j at index i

$$dp_{i,j} = \sum_{k=-1}^{1} dp_{i-1,j+k} = dp_{i-1,j-1} + dp_{i-1,j} + dp_{i-1,j+1}$$

The problem is that we need both $dp_{i-1,j-1}$ & $dp_{i-1,j+1}$ so how do we loop on the j dimension.

to get $dp_{i,j}$ we need
- $dp_{i-1,j-1}$
  - $dp_{i-2,j-2}$
  - $dp_{i-2,j-1}$
  - $dp_{i-2,j}$
- $dp_{i-1,j}$
- $dp_{i-1,j+1}$
  - $dp_{i-2,j}$
  - $dp_{i-2,j+1}$
  - $dp_{i-2,j+2}$