Knapsack Problem 2

CS 491 – Competitive Programming

Dr. Mattox Beckman

University of Illinois at Urbana-Champaign
Department of Computer Science

Fall 2023

Objectives

- Use a combination of divide and conquer to solve the knapsack-n problem.

Running Example

- Suppose we have the following items:

w	c
5	15
4	10
1	1

- Demonstrate that a greedy algorithm will not work.
  - Hint: Use a bag of capacity 13.

The intuition

- Suppose $f(S)$ is the best solution we can get from a bag of size S.
- Then, for some $0 < X < S$, we have $f(S) = f(S - X) + f(X)$.
- Do you belive that?

The intuition

- Suppose $f(S)$ is the best solution we can get from a bag of size S.
- Then, for some $0 < X < S$, we have $f(S) = f(S - X) + f(X)$.
- Do you belive that?

w = 5, c = 15	w = 4, c = 10	w = 4, c = 10

How to pick x

- Let W be the maximum weight item (in our case, 5).

$$\begin{aligned} S_1 = & \lfloor \tfrac{S-W}{2} \rfloor + (S - W) \mod 2 & = 4 \\ S_2 = & \lfloor \tfrac{S+W}{2} \rfloor & = 9 \end{aligned}$$

How to pick x

- Let W be the maximum weight item (in our case, 5).

$$
\begin{aligned}
S_1 &= \left\lfloor \frac{S-W}{2} \right\rfloor + (S-W) \mod 2 &= 4 \\
S_2 &= \left\lfloor \frac{S+W}{2} \right\rfloor &= 9
\end{aligned}
$$

w = 5, c = 15	w = 4, c = 10	w = 4, c = 10

How to pick x

- Let W be the maximum weight item (in our case, 5).

$$
\begin{aligned}
S_1 &= \lfloor \tfrac{S-W}{2} \rfloor + (S-W) \mod 2 &= 4 \\
S_2 &= \lfloor \tfrac{S+W}{2} \rfloor &= 9
\end{aligned}
$$

w = 5, c = 15	w = 4, c = 10	w = 4, c = 10

What we need to consider

- We need to consider all the pairs $(S_1.S_2), (S_1+1, S_2-1), \ldots (S_1+W/2, S_2-W/2)$.

$$\begin{aligned} f(13) = &\quad f(4)+f(9) = 10+20 = 30 \\ f(13) = &\quad f(5)+f(8) = 15+20 = 35 \\ f(13) = &\quad f(6)+f(7) = 16+17 = 33 \end{aligned}$$

- In this example, our best partition is at 5 and 8, giving us 35.

w = 5, c = 15	w = 4, c = 10	w = 4, c = 10

w = 5, c = 15	w = 4, c = 10	w = 4, c = 10

Bigger Example

- Remember the weights:

w	c
5	15
4	10
1	1

- What is the optimum cost for a bag of size 58?
  - $S_1 = 27, S_2 = 31$
  - So we need to compute $f(27) + f(31), f(28) + f(30), f(29) + f(29)$.

Bigger Example, Recurse!

- What is the optimum cost for a bag of size 53?
  - $S_1 = 27, S_2 = 31$
  - We will use recursion to compute $f(27), \cdots, f(31)$
- To compute $f(27) we have $S_1 = 11$.
- To compute $f(31) we have $S_2 = 18$.
  - We only compute S_1 for the left and S_2 for the right since we will need the whole range anyway.

Bigger Example, Recurse!

- What is the optimum cost for a bag of size 53?
  - $S_1 = 27, S_2 = 31$
  - We will use recursion to compute $f(27), \cdots, f(31)$
  - To compute $f(27)$ we have $S_1 = 11$.
  - To compute $f(31)$ we have $S_2 = 18$.
- Recurse again on 11 and 18…
  - $S_1 = 3, S_2 = 11$.
- The next recursion has $S_1 \leq 0$, so we stop here.

Base Case

- Compute up to $3W$ as a regular DP array.

```
for(long long i=0;i<N;i++) {
     cin >> w[i] >> c[i];
     maxw=max(maxw,w[i]);
 }
for (long long i=0;i<N;i++)
     for (long long j=w[i];j<=3*maxw;j++)
          dp[j]=max(dp[j],dp[j-w[i]]+c[i]);
```

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	1	2	3	10	11	12	13	20	21	22	23	30	31	32	33
0	1	2	3	10	15	16	17	20	25	30	31	32	35	40	45

Compute the S levels

```
e=0; S1=S; S2=S;
while(S1>0) {
     l[e]=S1; r[e]=S2;
     S1=(S1-maxw)/2+(S1-maxw)%2;
     S2=(S2+maxw)/2;
     e++;
}
```

For this case: we get

e	l	r
0	58	58
1	27	31
2	11	18
3	3	11

Recursively compute answers

```
for(long long i=e-1;i>=0; i--) {
    for (long long j=l[i];j<=r[i];j++) {
        if(j<=3*maxw)
            ans[i][j]=dp[j];
        else for(long long k=(j-maxw)/2+(j-maxw)%2;
                k*2LL<=j;
                k++)
            ans[i][j]=max(ans[i][j]
                    ,ans[i+1][k]+ans[i+1][j-k]);
    }
```

Recursively compute answers, e = 3

```
for(long long i=e-1;i>=0; i--) {
    for (long long j=l[i];j<=r[i];j++) {
        if(j<=3*maxw)
            ans[i][j]=dp[j];
        else for(long long k=(j-maxw)/2+(j-maxw)%2;
                    k*2LL<=j;
                    k++)
            ans[i][j]=max(ans[i][j]
                            ,ans[i+1][k]+ans[i+1][j-k]);
    }
```

- $ans[3][3..11] = 3, 10, 15, 16, 17, 20, 25, 30, 31$

Recursively compute answers, Rest of e

```
for(long long i=e-1;i>=0; i--) {
    for (long long j=l[i];j<=r[i];j++) {
        if(j<=3*maxw)
            ans[i][j]=dp[j];
        else for(long long k=(j-maxw)/2+(j-maxw)%2;
                k*2LL<=j;
                k++)
            ans[i][j]=max(ans[i][j]
                        ,ans[i+1][k]+ans[i+1][j-k]);
    }
```

- $ans[3][3..11] = 3, 10, 15, 16, 17, 20, 25, 30, 31$
- $ans[2][11..18] = 31, 32, 35, 40, 45, 46, 47, 50$
- $ans[1][27..31] = 77, 80, 85, 90, 91$
- $ans[0][58] = 170$

Optimization

- We don't need the whole array!!

```
for(long long i=e-1;i>=0; i--) {
    for (long long j=l[i];j<=r[i];j++) {
        if(j<=3*maxw)
            ans[i][j-l[i]]=dp[j];
        else for(long long k=(j-maxw)/2+(j-maxw)%2;
                k*2LL<=j;
                k++)
            ans[i][j-l[i]]=max(ans[i][j-l[i]]
                                    ,ans[i+1][k-l[i+1]] +
                                        ans[i+1][j-k-l[i+1]]);
    }
```