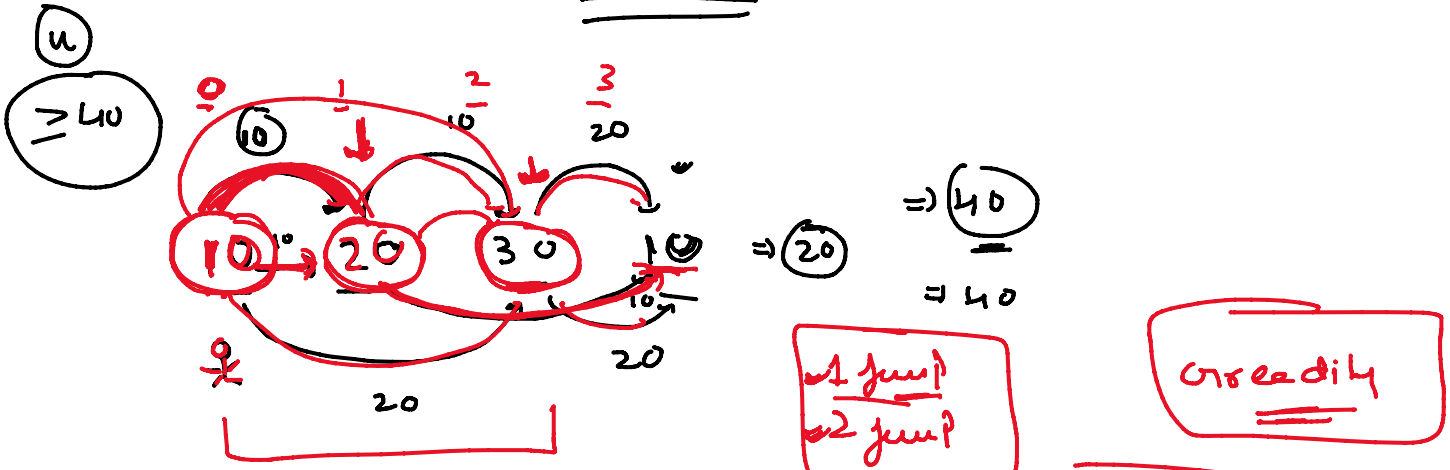
Class 80

09 May 2023 21:10

Class 80

Frog Jump



1 jump
2 jump

Greedily

Why Recursion !

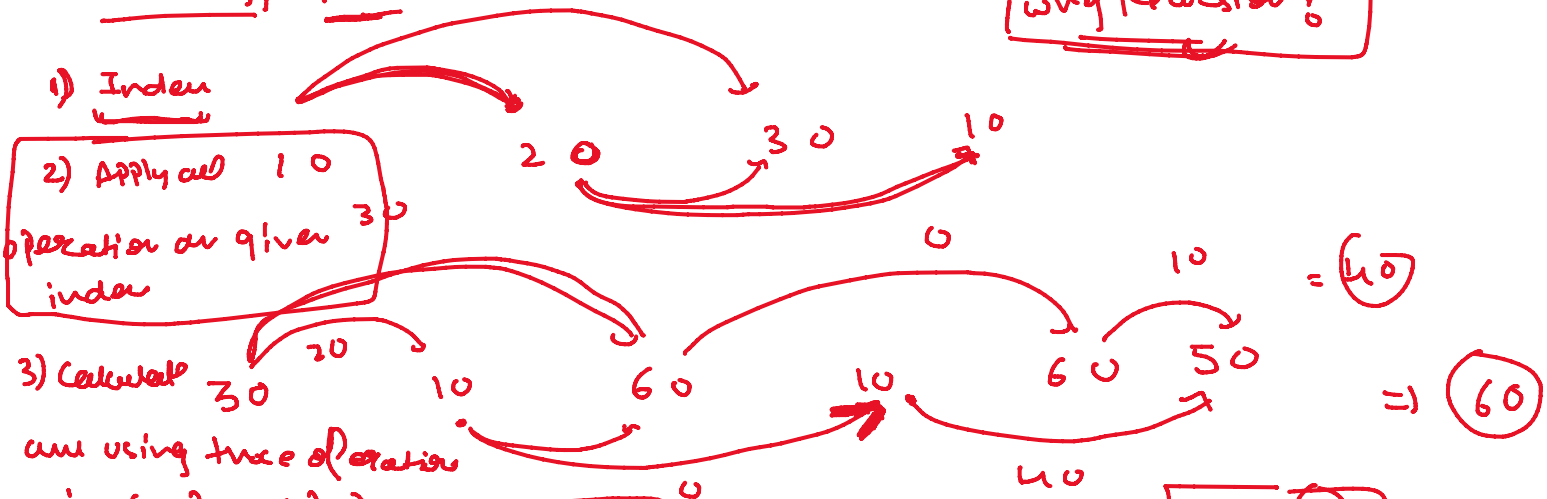
Min Energy spend

1) Index

2) Apply all 10 operation on given index

3) Calculate all using these operation

min (op1 , op2);



Recursion

0 → n

i → 0

```
int frogJump( vector<int>& energy, int idx, vector<int> dp
    if (idx == 0)
        return 0;
    if (idx == 1) return abs(energy[0] - energy[1])
    if(dp[idx] != -1) return dp[idx];
    int op1 = frogJump(energy, idx-1) + abs(energy[idx] - energy[idx-1])
```

0 - i^th index Min energy

int op2 = frogJump(energy, idx-2) + abs(energy[idx] - energy[idx-2]);

(-1)

dp[idx] =

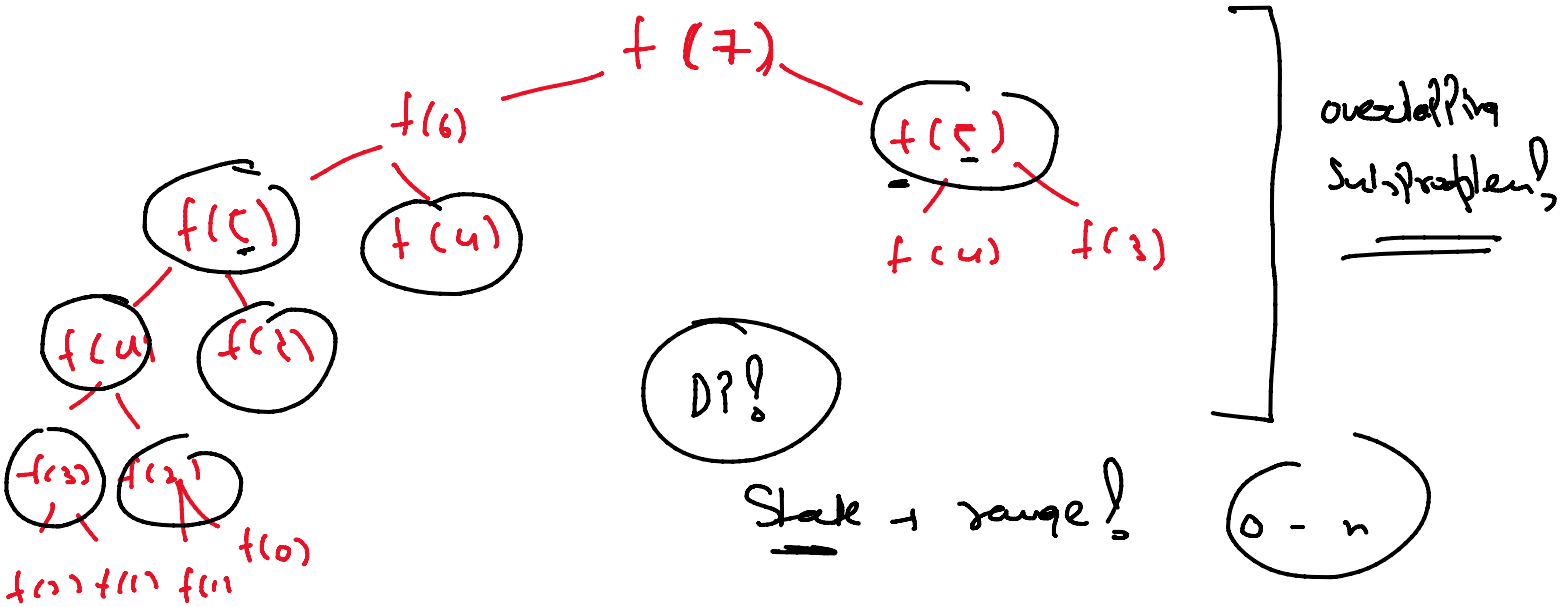
return min(op1, op2);

}

State => Min Max

State !

1 State, 1 single state



Memoization and Tabulation

vector<int> dp(n, -1);

frogJump(..., dp);

Tabulation

vector<int> dp(n, 0);

dp[0] = 0;

dp[1] = abs(energy(0) - energy(1));

int prev1, int prev2 , Space optimized version - O(1)

For (i=2; i<n; i++)

{

int op1 = dp[i-1] + abs(energy[i-1] - energy[i]);

int op2 = dp[i-2] + abs(energy[i-2] - energy[i]);

dp[i] = min(op1, op2);

}

i (i-1) (i-2)

return dp[n-1];

Frog jump P with K available jumps

K options

2

1 2 3 4 --- K

int frogjump(vector<int>& energy, int K, int idx)

{ if (idx == 0) return 0;

if (dp[idx] != -1) return dp[idx];

int mn = INT_MAX;

For (i=1; i<=K; i++)

if (idx - i >= 0)

{ mn = min(mn, frogjump(energy, K, idx - i) +

abs(energy[idx-i] - energy[i]);

}

return dp[idx] = mn

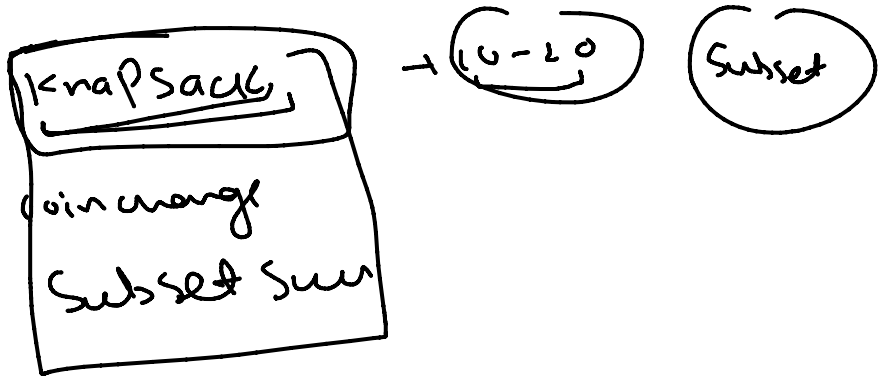
T.C = O(n * k)

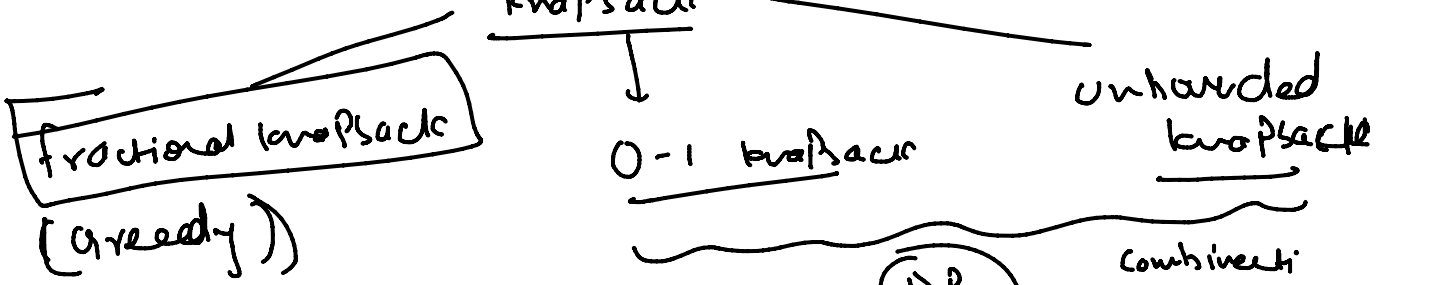
S.C = O(n)

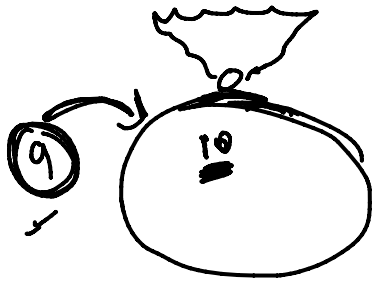
Tabulation

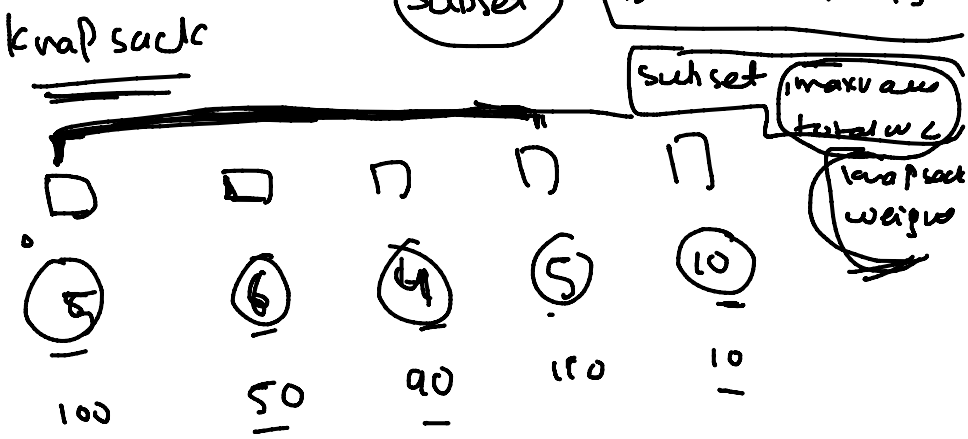
1D DP

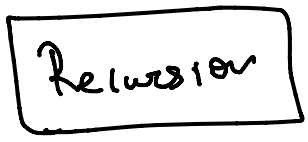
Count all ways











→ Index

→ Index

→ All options at an Index ✓
x

→ Max value which option → max(op1, op2)

subset,

0 1 2 -- n

int knap (vector<int> &weight, vector<int> &value, int idx, int maxweight, vector<vector<int>> &dp)

0 1 2 3 4

{ if (idx == 0)
return 0;

if (maxweight == 0)
return 0;

idx = 1
idx = 2
idx = 3

if (dp[idx][maxweight] != -1)
return dp[idx][maxweight];

int op1 = 0, int op2 = 0;

if (weight[idx-1] <= maxweight)

{ op1 = knap(weight, value, idx-1, maxweight - weight[idx-1]) + value[idx-1];

}

op2 = knap (weight, value, idx-1, maxweight);

dp[idx][weight] =

return max (op1, op2)

knap (10, n)

State 0 = 2

2 D vector

knap (8, n-1)

knap (8, n-2)

knap (8, n-2)

knap (10, n-1)

knap (8, n-2)

2D DP

size ⇒ (n+1)*(w+1) ~ 10^{12}

map

```
vector<vector<int>> dp(n+1, vector<int>(w+1, -1));
```

weight of knapsack

I level

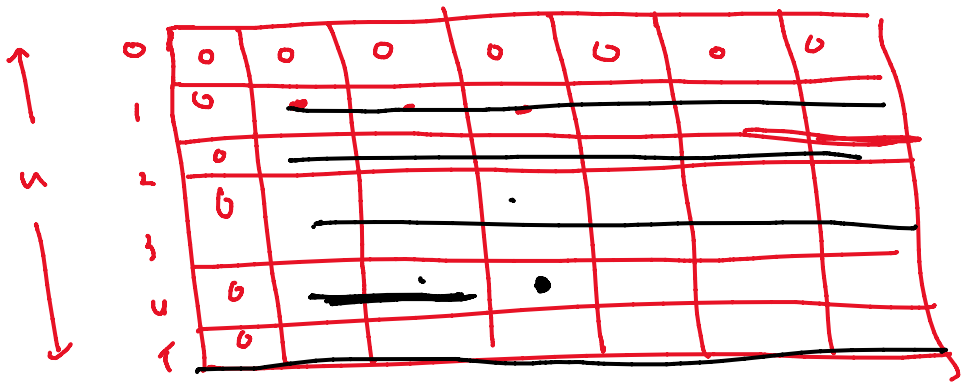
(idx, w)

10^{12}

100 , 1000

0 - (n+1)*(w+1)

Tabulation



Frog for P I, II

knapsack 0-1, fractional

unbounded

fibonacci, climbing stairs