(আগের পর্ব) আজকে আরো দুটি ক্লাসিকাল ডাইনামিক প্রোগ্রামিং প্রবলেম শিখবো। প্রথমটা হলো কয়েন চেঞ্জ। প্রবলেমটার নাম শুনেই বোঝা যাচ্ছে এটা টাকা ভাংতি করা নিয়ে প্রবলেম, তোমাকে সবথেকে কম সংখ্যক কয়েন ব্যবহার করে নির্দিষ্ট পরিমাণ টাকা ভাংতি করতে হবে।

মনে করো তোমার কাছে nn টা ভিন্ন ভিন্ন কয়েন আছে, কয়েনগুলোর ভ্যালুকে C_0,C_1…Cn−1C0,C1…Cn−1 দিয়ে প্রকাশ করা যায়। আর তোমাকে একটা অ্যামাউন্ট দেয়া আছে WW। এখন তোমাকে বলতে সর্বনিম্ন কয়টা কয়েন ব্যবহার করে তুমি WW অ্যামাউন্টটা বানাতে পারবে। প্রতিটা ভ্যালুর কয়েন আছে মাত্র ১টা করে।

একটা উদাহরণ দেখি। ধরা যাক কয়েনগুলোর ভ্যালু হলো C={2,5,9,13,15}C={2,5,9,13,15} টাকা। এখন তুমি এই কয়েনগুলো দিয়ে W=২২W=২২ বানাতে চাইলে একটা উপায় হলো 15+5+215+5+2, এক্ষেত্রে কয়েন লাগছে ৩টা। কিন্তু তুমি চাইলে ২টা কয়েন ব্যবহার করেও 2222 বানাতে পারো (9+13)(9+13)। আমাদের টার্গেট কয়েন ব্যবহার মিনিমাইজ করা।

অনেকে শুরুতে এটা গ্রিডী (greedy) পদ্ধতি সমাধানের চেষ্টা করে কিন্তু সেটা কাজ করবে না। তুমি যদি সবথেকে বড় কয়েন থেকে নেয়া শুরু করো তাহলে কয়েন সংখ্যা মিনিমাইজ নাও হতে পারে যেটা উপরের উদাহরণেও দেখেছি।

আমরা আগের মতোই একটা রিকার্সিভ ফর্মুলা তৈরি চেষ্টা করবো। প্রথমেই চিন্তা করি সাবপ্রবলেম বা স্টেট কি হবে। কয়েনগুলো কোন অর্ডারে নিতে হবে সেটা নির্দিষ্ট করা নেই, তবে আমরা সুবিধার জন্য ধরে নেই আমরা অ্যারের বাম পাশ থেকে কয়েন নেয়া শুরু করবো। তাহলে স্বাভাবিক ভাবেই মাথায় যে সাবপ্রবলেম আসবে সেটা হলো আমরা বর্তমানে কোন কয়েন নিয়ে কাজ করছি সেটা। তাহলে আমাদের রিকার্সিভ ফাংশন হবে f(i)f(i) । আমরা চেষ্টা করবো ii তম কয়েন থেকে n−1n−1 তম কয়েনগুলো দিয়ে WW বানাতে।

এখন ii তম কয়েন হাতে নিয়ে আমাদের দুইটা অপশন আছে:

- ii তম কয়েনটাকে ব্যবহার করা, তাহলে পরবর্তি সাবপ্রবলেম হবে f(i+1)f(i+1)।
- ii তম কয়েনটা ব্যবহার না করে পরের কয়েন (i+1)(i+1) এ চলে যাওয়া।, এবারও পরবর্তি সাবপ্রবলেম হবে f(i+1)f(i+1)।

এখন আমরা একটা ঝামেলায় পড়ে গেছি, যখন এক সাবপ্রবলেম থেকে আরেক সাবপ্রবলেমে যাচ্ছি তখন আমরা জানিনা আমাদের টার্গেট অ্যামাউন্ট এখন কত। শুরুতে টার্গেট অ্যামাউন্ট WW থাকলেও প্রতিবার কয়েন নিলে সেটা বদলে যাবে। আমাদেরকে কোনোভাবে সেটা মনে রাখতে হবে। মনে রাখার পদ্ধতি হলো আরেকটা স্টেট ব্যবহার করা। তাহলে আমরা নতুন করে ফাংশন ডিফাইন করি f(i,W)f(i,W) এবং অপশন দুটো নিয়ে আবার চিন্তা করি:

- ii তম কয়েনটাকে ব্যবহার করলে আমাদের টার্গেট বাকি থাকে W–C_iW–Ci। তাহলে পরবর্তি সাবপ্রবলেম হবে f(i+1,W–C_i)f(i+1,W–Ci)।
- ii তম কয়েনটাকে ব্যবহার না করলে আমাদের টার্গেট পরিবর্তন হবে না। তাহলে পরবর্তি সাবপ্রবলেম হবে f(i+1,W)f(i+1,W)।

আমাদেরকে দুইটা সাবপ্রবলেমের মিনিমাম না নিতে হবে। প্রথম অপশনে যেহেতু কয়েন নিচ্ছি তাই 1 যোগ করতে হবে। এবার তাহলে ফর্মুলাটা লিখে ফেলি:

$$f(i, W) = 0 \ if \ W = 0$$

$$f(i, W) = inf \ if \ W < 0$$

$$f(i) = min(1 + f(i + 1, W - C[i]), f(i + 1, W)) \ otherwise$$

এখন কোড লিখে ফেলা খুবই সহজ:

```
#define MAX_N 20
#define MAX_W 10000

#define INF 99999999
#define EMPTY_VALUE -1

int C[MAX_N];
int mem[MAX_N][MAX_W];
int n;

int f(int i, int W) {
    if (W < 0) return INF;
    if (i == n) {
        if (W == 0) return 0;
        return INF;
    }


    if (mem[i][W] != EMPTY_VALUE) {
        return mem[i][W];
    }

    int res_1 = 1 + f(i + 1, W - C[i]);
    int res_2 = f(i + 1, W);

    mem[i][W] = min(res_1, res_2);

    return mem[i][W];
}
```

কোনো অ্যামাউন্ট যদি বানানো না যায় তাহলে এই কোড INF রিটার্ন করবে।

কমপ্লেক্সিটি:

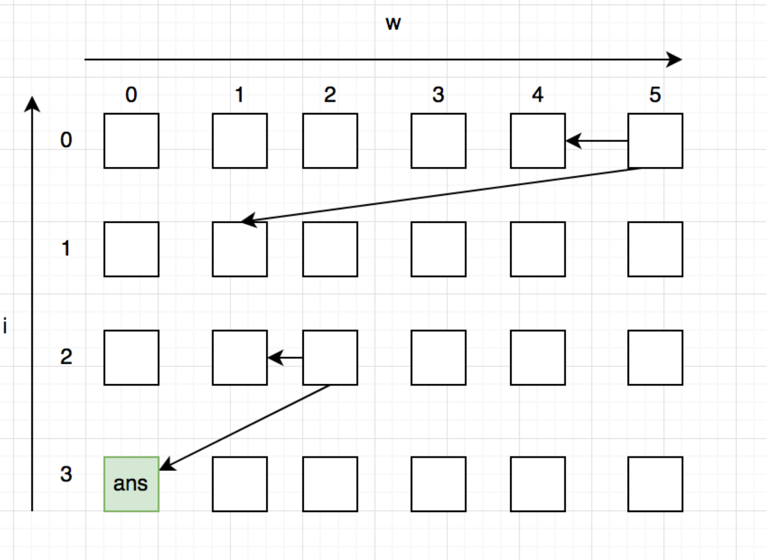
আমাদের সাবপ্রবলেম সংখ্যা n∗Wn∗W এবং প্রতিটা সাবপ্রবলেম অন্য সাবপ্রবলেম কল করা ছাড়া বাকি কাজ constant টাইমে করছি। টাইম কমপ্লেক্সিটি হবে O(nW)O(nW)। (এটা কিন্তু পলিনোমিয়াল কমপ্লেক্সিটি না, এটা সুডোপলিনোমিয়াল, তুমি গুগল করে এ বিষয়ে জেনে নিতে পারো)

ইটারেটিভ ভার্সন:

আমরা জানি ইটারেটিভ ভার্সনে যে সাবপ্রবলেমটা প্রথমে সলভ হচ্ছে সেখান থেকে টেবিল বিল্ডআপ শুরু করতে হয়। আমাদের ফর্মুলায় ii এর মান 00 থেকে সামনের দিকে যাচ্ছে এবং WW এর মান

পিছন দিকে যাচ্ছে। আমরা টেবিল বিল্ডআপ শুরু করবো $(n-1,0)(n-1,0)$ সেল থেকে। তাহলে প্রতিটা সাবপ্রবলেম টপোলজিকাল অর্ডারে আপডেট হবে। বুঝতে সমস্যা হলে আমার সাজেশন হবে

$n*Wn*W$ সাইজের একটা টেবিল খাতায় একে ঘরগুলো হাতে-কলমে আপডেট করা।



আমি একটা আর্বিট্রারি টেবিল একে দেখিয়ে দিলাম উত্তরটা কোন কোনায় থাকবে এবং সেলগুলো কোন ডিরেকশনে আপডেট হবে। তোমার কাজ হবে ঠিকমত আপডেট করা।

ইটারেটিভ ডিপিতে কর্নার কেস নিয়ে একটু সাবধান থাকতে হয় কারণ রিকার্সিভ ফাংশনের জায়গায় সরাসরি টেবিল থেকে আপডেট করছি, উল্টাপাল্টা ইনডেক্সে এক্সেস করলে রানটাইম এরোর দিবে। এখানে আমি যেটা করি, টেবিল এক্সেস করার কোডটুকু একটা নতুন ফাংশনের মাধ্যমে করি।

```
int evaluate_table(int i, int W, int n) {
    if (W < 0) return INF;
    if (i == n) {
        if (W == 0) return 0;
        return INF;
    }

    return mem[i][W];

}

int coin_change_iterative(int n, int target) {
    for (int i = n - 1;i >= 0;i--) {
        for (int w = 0; w <= target; w++) {

            int res_1 = 1 + evaluate_table(i + 1, w - C[i], n);
            int res_2 = evaluate_table(i + 1, w, n);
            mem[i][w] = min(res_1, res_2);
        }
```

```
    }

    return mem[0][target];
}
```

এখানে একটা দারুণ মেমরি অপটিমাইজেশনের সুযোগ আছে। খেয়াল করো, ভিতরের লুপে আমাদের খালি i+1i+1 রো এর ভ্যালুগুলো লাগছে। তারমানে ii এর মান যখন xx তখন খালি তোমার row x+1x+1 এর ভ্যালুগুলো দরকার হবে, বাকিগুলো টেবিল আর কোনো কাজে লাগবে না। তারমানে বর্তমান রো এবং তার আগের রো ছাড়া বাকিগুলো কাজে লাগছে না। এভাবে তুমি 2∗W2∗W সাইজের টেবিল ব্যবহার করেই প্রবলেমটা সলভ করতে পারো।

ভ্যারিয়েশন:

এবার আমরা কিছু ভ্যারিয়েশন দেখবো কয়েন চেঞ্জের। মনে করো এখনো আমাদের কয়েনের ভ্যালুগুলো একই আছে C={2,5,9,13,15}C={2,5,9,13,15}, কিন্তু এবার প্রতিটা কয়েনের সাপ্লাই অসীম। আগে 3030 বানাতে কয়েন লাগতো ৩টা (15+13+2)(15+13+2) কিন্তু এখন লাগবে ২টা (15+15)(15+15)।

এটা করা যাবে কোডে মাত্র ১টা ক্যারেক্টার মুছে দিয়ে। পরের লাইন পড়ার আগে একটু নিজে নিজে চিন্তা করো।

আগে আমরা একটা কয়েন নিলে পরের স্টেট f(i+1,W–C[i])f(i+1,W–C[i]) এ চলে যাচ্ছিলাম, তাই একটা কয়েন একবারের বেশি নেয়া হচ্ছিলো না। আমরা যদি ii তম কয়েন নেয়ার পর একই কয়েনে থেকে যাই, খালি WW আপডেট করি তাহলেই কিন্তু কাজ হয়ে যাবে। ফর্মুলাটা এবার হবে:

$$f(i) = min(1 + f(i, C - W[i]), f(i + 1, W))$$

আগেরটার সাথে পার্থক্য হলো প্রথম i+1i+1 টাকে ii করে দিয়েছি।

```
#define MAX_N 20
#define MAX_W 10000

#define INF 99999999
#define EMPTY_VALUE -1

int C[MAX_N];
int mem[MAX_N][MAX_W];
int n;

int f(int i, int W) {
    if (W < 0) return INF;
    if (i == n) {
        if (W == 0) return 0;
        return INF;
    }

    if (mem[i][W] != EMPTY_VALUE) {
        return mem[i][W];
    }

```

```
    int res_1 = 1 + f(i, W - C[i]); //only this line updated
    int res_2 = f(i + 1, W);

    mem[i][W] = min(res_1, res_2);

    return mem[i][W];
}
```

এখন আমাদের আরেকটু অপটিমাইজেশনের জায়গা তৈরি হয়েছে, আমরা চাইলে স্টেট কমিয়ে স্পেস বাচাতে পারি। এখন যেহেতু আমাদের আনলিমিটেড কয়েন আছে, আমরা কোন কয়েন কয়বার নিচ্ছি সেটা নিয়ে আর চিন্তা করতে হচ্ছে না। আমরা চাইলে সাবপ্রবলেমকে খালি f(W)f(W) দিয়ে ডিফাইন করতে পারি। আমরা বের করতে চাই WW বানাতে কয়টা কয়েন লাগবে, কোন কয়েন আমি এখন নিবো সেটা ব্যাপার না। এরপর ফাংশনের ভিতরে একটা লুপ চালিয়ে একটা একটা করে কয়েন নিয়ে চেষ্টা করবো।

$$f(W) = 0 \; if W = 0$$

$$f(W) = inf \; if \; W < 0$$

$$f(W) = min(\,f(W - C_i)\,)\, for\; i \in [0, n)$$

এবার টাইম কমপ্লেক্সিটি একই থাকলেও টেবিলের সাইজ অনেক কমে যাবে। **স্টেট কমিয়ে দিয়ে লুপ ব্যবহার করে মেমরি কমানো ডাইনামিক প্রোগ্রামিং এর খুবই কমন একটা ট্রিক।**

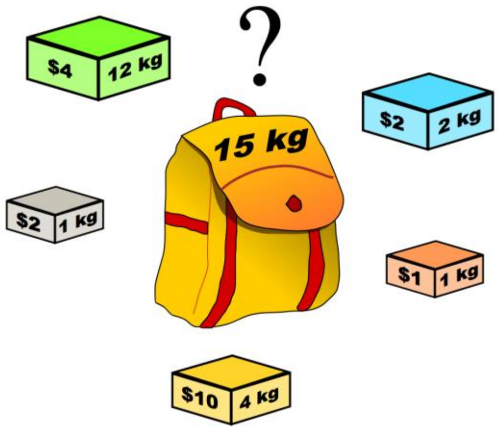
```
#define MAX_N 20
#define MAX_W 10000

#define INF 99999999
#define EMPTY_VALUE -1

int C[MAX_N];
int mem[MAX_W];
int n;

int f_optimized(int W) {
    if (W < 0) return INF;
    if (W == 0) return 0;

    if (mem[W] != EMPTY_VALUE) {
        return mem[W];
    }

    int ans = INF;
    for (int i = 0;i < n;i++) {
        ans = min(ans, 1 + f_optimized(W - C[i]));
    }

    mem[W] = ans;
    return mem[W];
}
```

এখন একটা বাড়ির কাজ: যদি বলা হয় প্রতিটা কয়েন সর্বোচ্চ kk বার ব্যবহার করা যাবে তাহলে কি করবে?

০-১ ন্যাপস্যাক:

তুমি কয়েন চেঞ্জ বেশ কয়েকভাবে সলভ করতে শিখে গিয়েছো, তাই 0-1 ন্যাপস্যাক আমি সলভ করে দিব না। আমি প্রবলেমটা কি সেটা বলবো এবং কিছু হিন্টস দিবো।

ন্যাপস্যাক এর বাংলা হলো থলে বা ব্যাগ। তোমার কাছে একটা ব্যাগ আছে যারা নির্দিষ্ট একটা ক্যাপাসিটি আছে, ধরলাম সেই ক্যাপাসিটি হলো CC। এখন তোমার সামনে nn টা আইটেম আছে, প্রতিটা আইটেমের নির্দিষ্ট দাম এবং ওজন আছে। নিচের ছবিটা wikipedia থেকে নেয়া:



এখন তোমাকে বলতে হবে তুমি সর্বোচ্চ কত দামের জিনিস ব্যাগে ভরতে পারবে। “০-১” ন্যাপস্যাক বলার কারণ হলো কোনো জিনিস নিলে পুরোটাই নিতে হবে, ভেঙে অর্ধেক করে নিতে পারবে না। ভেঙে নেয়ার নিয়ম থাকলে সেই প্রবলেমটাকে ফ্র্যাকশনাল ন্যাপস্যাক বলে। ফ্র্যাকশনালের ক্ষেত্রে দামি জিনিসগুলো আগে নিলেই অপটিমাল রেজাল্ট পাওয়া যায়, ০-১ ন্যাপস্যাক এ সেটা কাজ করবে না।

এখানে ইনপুট হিসাবে দেয়া হবে দুটি অ্যারে PP এবং WW। ii তম বস্তুর দাম P_iPi এবং ওজন W_iWi।

আমাদের সাবপ্রবলেম আগের মতোই হবে f(i,C)f(i,C) যা দিয়ে বুঝাবে ii থেকে শুরু করে $n-1n-1$ তম আইটেম গুলো দিয়ে পাওয়া সর্বোচ্চ প্রফিট। সেখান থেকে আমাদের দুইটা চয়েস

- ii তম আইটেম ব্যাগে না ভরা, তাহলে পরবর্তী সাবপ্রবলেম হবে f(i+1,C)f(i+1,C)
- ii তম আইটেম নেয়া, পরবর্তি সাবপ্রবলেম হবে $f(i+1,C–W_i)f(i+1,C–Wi)$।

আমাদেরকে এই দুইটার মধ্যে থেকে বড়টাকে নিতে হবে। ২য় ক্ষেত্রে প্রফিট হবে P_iPi, সেটাও যোগ করতে হবে।

$$f(i, C) = max(f(i+1,\ C),\ f(i+1,\ C - W_i) + P_i)$$

তোমার কাজ হবে এই ফর্মুলার রিকার্সিভ এবং ইটারেটিভ ভার্সন লেখা এবং ইটারেটিভ ভার্সনের মেমরি অপটিমাইজ করা। আজ এই পর্যন্তই। তুমি ডাইনামিক প্রোগ্রামিং এর অনেকগুলো ট্রিকস শিখে ফেলেছো এরমধ্যেই, এখন বেশি করে প্রবলেম সলভ করলে জিনিসটা আয়ত্বে এসে যাবে। তুমি এখন পর্যন্ত যেসব প্রবলেম দেখেছো সেগুলোতে রেজাল্ট ম্যাক্সিমাইজ বা মিনিমাইজ করতে হয়, এরপর আমরা দেখবো কম্বিনেটরিক্সের কিছু প্রবলেম।

প্র্যাকটিস প্রবলেম:

https://leetcode.com/problems/coin-change/
https://leetcode.com/problems/minimum-cost-for-tickets/
https://www.spoj.com/problems/KNAPSACK/