What is Dynamic Programming?

Method for solving optimization problems by breaking a problem into smaller subproblems.

What is an optimization problem?

Enumeration

- You want to climb n steps. At any step, you can either climb 1, 2 or 3 steps. How many different ways can you climb the n stairs?

Minimization/Maximization

- We have an array A[1..N] of stock prices at different days. We want to buy and sell at 2 different days. Find the maximum amount of money we can make.

Yes/No Questions

- Can you form 10 dollars with x 5 pennies, y 10 pennies, and z quarters?

Problem 1: Stairs Climbing

You can climb 1 or 2 stairs with one step. How many different ways can you climb n stairs?

Naive solution:

Let f(n) be the number of different ways to climb n stairs.

How can we reach the nth stair??

(1) Be at the (n-2)th stair, then climb 2 steps

(2) Be at the (n-1)th stair, then climb 1 step.

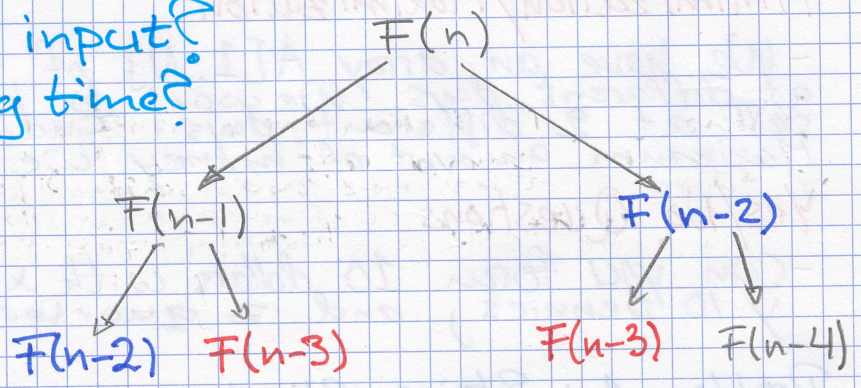
For (1), number of ways to reach the (n-2)th stair is f(n-2)

For (2), number of ways to reach the (n-1)th stair is $f(n-1)$

$f(n) = f(n-1) + f(n-2)$. Use recursion!
(Base cases are $f(1) = 1$, $f(2) = 2$).

```
int stairs(int n){
    if (n == 1) return 1;
    if (n == 2) return 2;
    return stairs(n-1) + stairs(n-2);
}
```

Large input?
Running time?



Solution? Dynamic Programming!

Let S[1...n] be an empty array of size n
S[1] = 1
S[2] = 2
For i = 3 to n:
    S[i] = S[i-1] + S[i-2]
Return S[n]

```
int n = 45;
int* S = new int[n+1];
S[1] = 1;
S[2] = 2;
for (int i=3; i<=n; ++i){
    S[i] = S[i-1] + S[i-2];
}
cout << S[n] << endl;
```

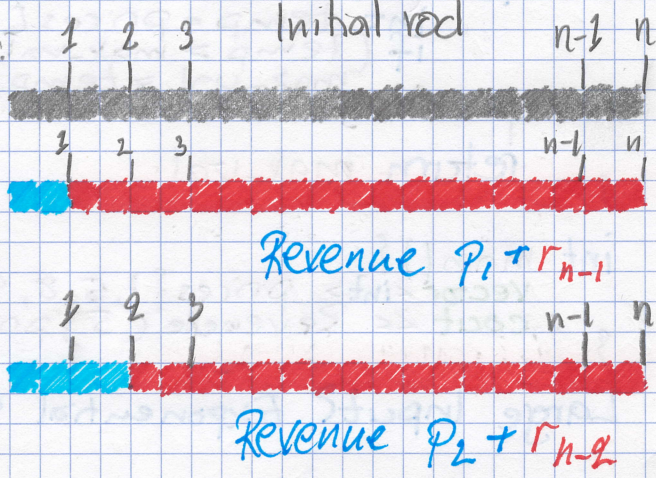
Problem 2: Cutting Rods

Given a rod of length n and prices $P[i]$ for $i = 1 \ldots n$, where $P[i]$ is the price of a rod of length i. Find the maximum total revenue you can make by cutting and selling the rod. (Assume no cost for cutting the rod.)

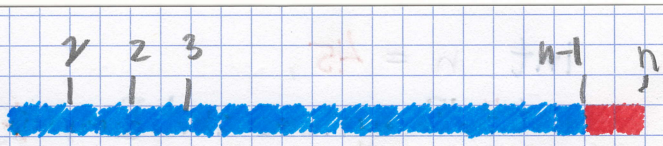
length i	1	2	3	4	5	6	7	8	9	10
price p_i	1	5	8	9	10	17	17	20	24	30

Recursive solution:

- Cut a piece of length 1
- Find the optimal division for length $n-1$
- Cut a piece of length 2
- Find the optimal division for length $n-2$



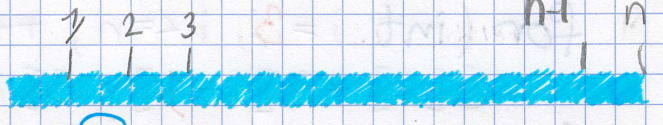
- Cut a piece of length $n-1$
- Find the optimal division for length n



Revenue $p_{n-1} + r_1 =$

$p_{n-1} + p_1$

Sell in one piece



Revenue p_n

The best choice is the maximum of

$p_1 + r_{n-1}, p_2 + r_{n-2}, \dots, p_{n-1} + r_1, p_n$

Define $R[n]$ as the maximum revenue you can make from a rod of length n. Then

$R[n] = \max\{P[1]+R[n-1], P[2]+R[n-2], P[3]+R[n-3], \dots, P[n-1]+R[1], P[n]+R[0]\}$

Base case $R[0] = 0$

```
int Revenue(int n, vector<int>& prices){
    if (n==0) return 0;
    int max_val = -1;
    for (int i=0; i<n; ++i){
        int temp = prices[n-i-1] + Revenue(i, prices);
        if (temp > max_val){
            max_val = temp;
        }
    }
    return max_val;
}

int main(){
    vector<int> prices{1,5,8,9,10};
    cout << Revenue(5, prices);
}
```

Large Input? Exponential Solution.

Solution? Dynamic Programming!

Let R[0..n] be an empty array of size n

```
R[0] = 0
For i = 1 to n
    max_val = -1
    For j = 1 to i:
        temp = prices[j] + R[i-j]
        if (temp > max_val)
            max_val = temp
    R[i] = max_val
Return R[n]
```

$O(n^2)$

```
vector<int> R(n+1);
R[0] = 0;
for (int i = 1; i <= n; ++i) {
    int max_val = -1;
    for (int j = 1; j <= i; ++j) {
        int temp = prices[j-1] + R[i-j];
        if (temp > max_val) {
            max_val = temp;
        }
    }
    R[i] = max_val;
}
return R[n];
```

Problem 3: House Robber

You are a professional robber planning to rob houses along a street. Each house has a certain amount of money stashed, the only constraint stopping you from robbing each of them is that adjacent house have security system connected and it will automatically contact the police if two adjacent houses were broken into on the same night.

Given a list of non-negative integers representing the amount of money of each house, determine the maximum amount of money you can rob tonight without alerting the police.

A[1..5] = {1, 2, 3, 4, 5}
Can rob 2, 4
Can rob 1, 3, 5
Can rob 1, 3
Can rob 2, 4
...

Define R[i] as the maximum money you can steal from house 1 to house i

There is two cases:

1) You steal house i
2) You Don't steal house i.

If you steal house i, then you can't steal house i-1. But can steal from house 1, 2, ..., (n-2). So if you steal from house i, then

$$R[i] = \underbrace{nums[i]}_{money} + \max\left(R[i-2], R[i-3], \ldots, R[1]\right)$$

If you don't steal from house i, then the max money you can steal is R[i-1]

$$R[i] = \max\left(R[i-1], nums[i]+R[i-2], nums[i]+R[i-3], \ldots, nums[i]+R[1]\right)$$

Base case $R[1] = nums[1]$

Possible Recursive code

```
int Rob(int i, vector<int>& nums){
    if (i == 0) return nums[0];
    int max_val = -1;
    max_val = max(max_val, Rob(i-1, nums));
    for (int j = i-2; j >= 0; --j){
        max_val = max(max_val, Rob(j, nums)+nums[i-1]);
    }
    return max_val;
}
int main(){
    vector<int> nums{1,2,3,4,5};
    cout << Rob(5, nums);
}
```

R[i-1]

Large Input? Exponential Solution!

Solution? Dynamic Programming!

Let R[0..n] be an array of size n+1

S[0] = nums[0]

S[1] = max(nums[0], nums[1])

For i = 2 to n:

    R[i] = R[i-1]

    For j = 0 to i-2:

        R[i] = max(R[i], nums[i] + R[j])

Return R[n]

$O(n^2)$

R[i] = max(R[i-1], nums[i] + R[i-2])

```
int rob(int n, vector<int>& nums){
    vector<int> R(n);
    R[0] = nums[0];
    R[1] = max(nums[0], nums[1]);
    for (int i=2; i<n; ++i){
        R[i] = R[i-1];
        for (int j=i-2; j>=0; --j){
            R[i] = max(R[i], R[j]+nums[i]);
        }
    }
    return R[n-1];
}
```

Problem 4: Best Time to Buy and Sell Stock

Say you have an array for which the ith element is the price of a given stock on day i.

If you were only permitted to complete at most one transaction (ie, buy one and sell one share of the stock), design an algorithm to find the maximum profit.

Examples

Input: [7, 1, 5, 3, 6, 4]

Output: 5

max. difference = 6-1 = 5 (not 7-1=6, as selling price needs to be larger than buying price)

Input: [7, 6, 4, 3, 1]

Output: 0

In this case, no transaction is done, ie, max profit = 0.

Recursive Solution

Define R[i] as the maximum amount of money you can earn from day 1 to day i.

There are two cases.

1) Sell at day i

2) Not Sell at day i

If You Sell at day i, then you can find the minimum value in days 1 to i so that you make maximum profit. so

$$R[i] = Price[i] - \min(Price[i], Price[i-1], \ldots, Price[1])$$

If you don't sell at day i, then the maximum amount of money you can make is R[i-1].

$$R[i] = \max(R[i-1], Price[i] - \min(Price[i], Price[i-1], \ldots, Price[1]))$$

Base case R[1] = 0

Recursive Solution Code

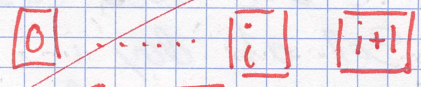
```
int R(int i, vector<int>& P){
    if(i==0) return 0;
    int max_val = R(i-1, P);
    for(int j=1; j<=i; ++j){
        max_val = max(max_val, P[i-1]-P[j-1]);
    }
    return max_val;
}

int main(){
    vector<int> P1{7, 1, 5, 4, 6, 4},
                P2{7, 6, 4, 3, 1};
    cout << R(6, P1) << endl << R(5, P2);
}
```

Time Complexity? $O(n^2)$ like naive Solution

Solution: Dynamic Programming

Let R[0..n-1] be an empty array of size n

```
R[0] = 0
For i = 1 to n-1:
    R[i] = R[i-1]
    For j = 0 to i:
        R[i] = max(R[i], P[i] - P[j])
Return R[n-1]
```



$O(n^2)$

Optimized Dynamic Programming!

Let R[0..n-1] be an empty array of size n

```
R[0] = 0
Min_value = P[0]
For i = 1 to n-1:
    Min_value = min(Min_value, P[i])
    R[i] = max(R[i-1], P[i] - Min_value)
Return R[n-1]
```

$O(n)$ Solution!

Problem 5: Maximum Length of Pair Chain

You are given n pairs of numbers. In every pair, the first number is always smaller than the second number.

Now, we define a pair (c,d) can follow another pair (a,b) if and only if $b < c$. Chain of pairs can be formed in this fashion.

Given a set of pairs, find the length longest chain which can be formed. You needn't use up all the given pairs. You can select pairs in any order.

Example

Input: [[1,2], [2,3], [3,4]]

Output: 2

Explanation: The longest chain is $[\overset{a}{1},\overset{b}{2}] \rightarrow [\overset{c}{3},\overset{d}{4}]$

Recursive Solution!

First note that if (a,b) is before (c,d) then:

$$a < b < c < d$$

For example $[1,2] \rightarrow [3,4]$, $1 < 2 < 3 < 4$

Sort all the pairs.

After sorting, Define R[i] as the longest chain from pair 1 to pair i

Two cases:

1) The longest chain ends in pair i

2) The longest chain does not end in pair i

For 1), then

$R[i] = \max(1+R[a], 1+R[b], 1+R[c], \ldots)$,

where a, b, c fit ith pair

For 2), then $R[i] = R[i-1]$

Combining both, we get a recurrence for all cases:

$R[i] = max(R[i-1], 1+R[a], 1+R[b], \ldots)$

Base case $R[1] = 1$

Recursive Solution / DP:

Works, but exponential! So, we use dynamic programming.

Let R[0..n-1] be an empty array of size n

Sort (pairs) $\Theta(n \log n)$

R[0] = 1

For i = 1 to n-1

R[i] = R[i-1]

For j = 0 to i-1:

if (pairs[j].second < pairs[i].first)

R[i] = max(R[i], R[j]+1)

Return R[n-1]

```
int findLongestChain(vector<vector<int>>& pairs){
    vector<int> dp(pairs.size());
    sort(pairs.begin(), pairs.end());
    dp[0] = 1;

    for(int i=1; i<pairs.size(); ++i){
        dp[i] = dp[i-1];
        for(int j=i-1; j>=0; --j){
            if(pairs[j][1] < pairs[i][0]){
                dp[i] = max(dp[i], dp[j]+1);
            }
        }
    }
    return dp[pairs.size()-1];
}
```

```
int main(){
    vector<int> C1{1, 2}, C2{2,3}, C3{3,4};
    vector<vector<int>> pairs{C1, C2, C3};
    cout << findLongestChain(pairs);
}
```

Problem 6: 0/1 Knapsack

A set of n items, where item i has weight w[i] and value v[i], and a knapsack with capacity W.

Suppose to pick a "few" elements from the n elements such that their weight is less than or equal to W, but their summed value is maximized.

Example:

$W = 8$, $n = 3$

$w[1] = 2$, $v[1] = 10$

$w[2] = 5$, $v[2] = 12$

$w[3] = 8$, $v[3] = 21$

Optimum is choose 1 of object 1 & of object 2, for total weight of $2+5 = 7$ and total value $10+12 = 22$. This is maximum possible value with 8 weight.

Attempt #1:

Define $R[j]$ to be the longest obtainable value for a knapsack with capacity j

$$R[j] = \max(0, v[1] + R[j-w[1]], v[2] + R[j-w[2]], \ldots, v[n] + R[j-w[n]])$$

Base case $R[j] = 0$, $j <= 0$

Wrong solution! Because we might pick more than once from the same item!

Have to change our definition of R to take this into consideration

Attempt #2

Define R[i][j] to be the largest obtainable value for a knapsack with capacity j using the first i element only

2-Dimensional Dynamic Programming

Two cases:

1) Use the i^{th} element
2) Don't use the i^{th} element

For 1) we have

$$R[i][j] = v[i] + R[i-1][j-w[i]]$$

For 2) we have

$$R[i][j] = R[i-1][j]$$

$$R[i][j] = \max\left(v[i] + R[i-1][j-w[i]], R[i-1][j]\right)$$

Base case: $R[i][j] = 0$ if $i = 0$ or $j = 0$

The answer to our problem is $R[n, w]$.

Dynamic Programming Pseudocode:

Let R[0..n, 0..W] be a new 2D array all zeros.

for i = 1 to n

  for j = 1 to W

    if (j-w[i] >= 0 and v[i] + R[i-1][j-w[i]] > R[i-1][j])

      R[i][j] = v[i] + R[i-1][j-w[i]]

```
        else
            R[i][j] = R[i-1][j]
Return R[n][W]
```

```
int Knapsack(int n, int W, vector<int> w, vector<int> v){
    vector<vector<int>> R(n+1, vector<int>(W+1, 0));
    for(int i=1; i<=n; ++i){
        for(int j=1; j<=W; ++j){
            if(j-w[i]>=0 && v[i-1]+R[i-1][j-w[i-1]]>
               R[i-1][j]){
                R[i][j] = v[i-1]+R[i-1][j-w[i-1]];
            }
            else{
                R[i][j] = R[i-1][j];
            }
        }
    }
    return R[n][W];
}

int main(){
    vector<int> weights{5,4,6,3},
                values{10,40,30,50};
    cout << Knapsack(4, 9, weights, values);
}
```

Problem 7: Longest Common Subsequence

Given two sequences $X[1..m]$ and $Y[1..n]$ find the longest common subsequence between them.

Example:

X: A **B** A **C** **B** D **A** B

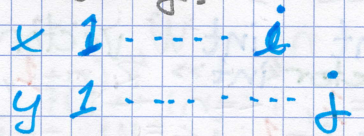
Y: **B** D **C** A **B** **A**

Longest Common Subsequence: BCBA

Has several applications in Genetics.

Recursive Solution

Define C[i][j] to be the length of the longest common subsequence of X[1..i] and Y[1..j]



The answer to our question is C[m][n]

There are two cases:

1) X[i] = Y[j]
2) X[i] != Y[j]

In 1) X[i] and Y[j] must be in the longest common subsequence

In 2) the longest common subsequence is either in X[1..i] and Y[1..j-1] or X[1..i-1] and Y[1..j]

$$c[i,j] = \begin{cases} 0 & \text{if } i=0 \text{ or } j=0 \\ c[i-1,j-1]+1 & \text{if } i,j>0 \text{ and } x_i = y_j \\ \max(c[i,j-1], c[i-1,j]) & \text{if } i,j>0 \text{ and } x_i \neq y_j \end{cases}$$

Dynamic Programming Pseudocode

```
Let C[0..m,0..n] be a new 2D array of all 0
For i=1 to m
    For j=1 to n
        if (X[i]=Y[j])
            C[i][j]=C[i-1][j-1]+1
        else
            C[i][j]=max(C[i-1][j], C[i][j-1])
Return C[m][n]
```

```
int LCSubsequence(string s, string t){
    int m = s.size();
    int n = t.size();
    vector<vector<int> > C(m+1, vector<int> (n+1, 0));
    for(int i=1; i<=m; ++i){
        for(int j=1 ; j<=n ; ++j){
            if (s[i-1] == t[j-1]){
                C[i][j] = C[i-1][j-1] +1;
            }
            else {
                C[i][j] = max(C[i-1][j], C[i][j-1]);
            }
        }
    }
    return C[m][n];
}

int main(){
    string s= "ABACBDAB";
    string t= "BDCABA";
    cout<< LCSubsequence(s, t);
}
```

Problem 8: Longest Common Substring

Given two strings X[1..m] and Y[1..n], we want to find the longest common substring between the two.

Example

X: D E A D B E E F

Y: E A T B E E F

Two common substrings are EA and BEEF, the longest is BEEF with length 4. So return 4.

Recursive Solution

Let d[i][j] be the length of the longest common substring of X[1..i] and Y[1..j] that ends at X[i] and Y[j].

We are changing the problem?

Yes, but then the optimal solution is max(d[i][j]) for all i,j!

Two cases:

1) X[i] = Y[j]
2) X[i] != Y[j]

For 1), then the LCS of X[1..i] and Y[1..j] is just LCS of X[1..i-1] and Y[1..j-1], plus X[i]=Y[j]

For 2), then there can't be a common substring ending at X[i] and Y[j]! So the answer is 0.

$$d[i,j] = \begin{cases} d[i-1,j-1]+1 & \text{if } x_i = y_j \\ 0 & \text{if } x_i \neq y_j \end{cases}$$

Dynamic Programming Pseudocode

```
Let d[0..m, 0..n] be a 2D array of all 0s
Max_value = -1
For i = 1 to m
    For j = 1 to n
        if (x[i] == y[j])
            d[i][j] = 1 + d[i-1][j-1]
        else
            d[i][j] = 0

        Max_value = max(Max_value, d[i][j])
Return Max_value
```

```
int LCS(string s, string t){
    int m = s.size();
    int n = t.size();
    int max_val = -1;
    vector<vector<int>> d(m+1, vector<int>(n+1, 0));
    for(int i = 1; i <= m; ++i){
        for(int j = 1; j <= n; ++j){
            if (s[i-1] == t[j-1]){
                d[i][j] = d[i-1][j-1] + 1;
            }else{
                d[i][j] = 0;
            }
            max_val = max(max_val, d[i][j]);
        }
    }
    return max_val;
}
int main(){
    string s = "DEADBEEF", t = "EATBEEF";
    cout << LCS(s,t);
}
```

Problem 9: Maximum Sum of a Contiguous Subarray problem

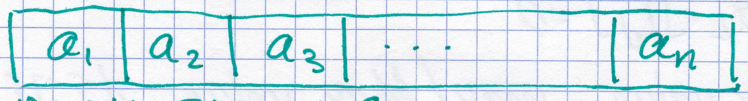
Find the contiguous subarray within an array (containing at least one number) which has the largest sum.

For example, given the array

$[-2, 1, -3, 4, -1, 2, 1, -5, 4]$

the contiguous subarray $[4, -1, 2, 1]$ has the largest sum $= 6$

$V = [a_1, a_1+a_2, a_1+a_2+a_3, \ldots]$ $\quad V[j] - V[i]$



max_sum ← 0

(i, j) $\quad a_i$ ——— a_j $\quad$ current_sum

```
max_sum = 0
for i ← 1 to n:
    for j ← i+1 to n:
        current_sum ← 0
        for k ← i to j:
            current_sum = current_sum + a_k
        max_sum ← max(max_sum, current_sum)
return max_sum
```

$O(n^3)$

$O(n^2)$

$O(n^2) \rightarrow O(n)$

V[i] as max sum of a contiguous subarray ends at the i^{th} element

$a_1 \; a_2 \; a_3 \; [a_j \cdots a_i]$

$\text{Sum}([a_{j-1} \ldots a_i]) < \text{Sum}(a_j \cdots a_i) = V[i]$

$a_1 \quad a_2 \quad a_3 \quad a_4 [a_k \, a_5 \, \ldots \, a_{i-1} \, \underline{a_i}]$

$\exists k$ s-t $V[i] = \text{Sum}(a_k \ldots a_i)$

$k < i \quad V[i] = \text{Sum}(a_k \ldots a_{i-1}) + a_i$

$k = i \quad V[i] = a_i$

$\Theta(1 \le i \le n)$

$V[i] = \max(a_i, V[i-1] + a_i)$ $\Theta(1)$

$\Theta(n) \cdot \Theta(1) = \Theta(n)$ time!

$V[i]$ max sum of Cont. Subarray from element $1 \ldots i$ But ending at a_i

Solution $= V[n] = \max(V[1], V[2], \ldots V[n])$

$V[i]$ max Contiguous subarray ending at element a_i

$V[i] = \max(V[i-1] + a_i, a_i)$

End solution: $\max(V[1], \ldots, V[n])$

Let $V[1..n]$ be an empty array

$V[1] = a_1$

for $i \leftarrow 2$ to n: $\Theta(n)$

$V[i] = \max(V[i-1] + a_i, a_i)$ $\Theta(1)$

return max_elem of V $\Theta(n)$

CORRECTION: House robber Problem with the fixed base cases

$R[i] = \max(R[i-1], nums[i] + R[i-2], nums[i] + R[i-3], \ldots, nums[i] + R[1])$

Base case $R[1] = nums[1]$

$R[2] = \max(nums[1], nums[2])$

```
int Rob(int i, vector<int>& nums){
    if (i==1) return nums[0];
    if (i==2) return max(nums[0], nums[1]);
    int max_val = -1;
    max_val = max(max_val, Rob(i-1, nums));
    for(int j=i-2; j>0; --j){
        max_val = max(max_val, Rob(j, nums) +
                              nums[i-1]);
    }
    return max_val;
}
int main(){
    vector<int> nums{1,2,3,4,5};
    cout << Rob(5, nums) << endl;
    vector<int> nums2{1,2,3,4,5,6};
    cout << Rob(6, nums2) << endl;
    return 0;
}
```

Large Input? Exponential Solution!