Dynamic Programming

- Recursion
  + Memoization (= remembering, keeping in memory).
- Systematic implementation of a brute-force strategy.

Def Fibonacci Numbers

F_0	F_1	F_2	F_3	F_4	F_5	F_6	...
0	1	1	2	3	5	8	...

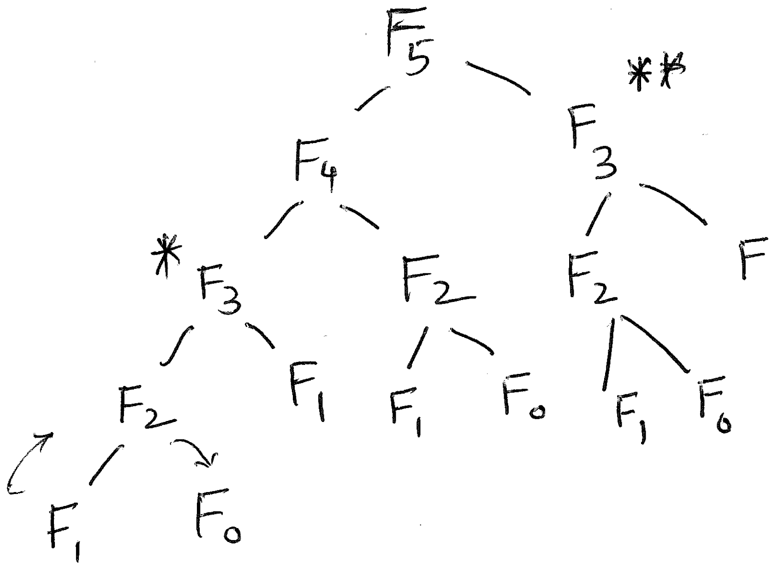
- $F_0 = 0$, $F_1 = 1$, $F_i = F_{i-1} + F_{i-2}$ $\forall i \geq 2$.

Problem Given n, compute F_n.

Recursive algo

```
int F(n) {
    if n=0, return 0.
    if n=1, return 1.
    Else,  a = F(n-1).
           b = F(n-1).
           return a+b.
}.
```

Unrolling recursion.



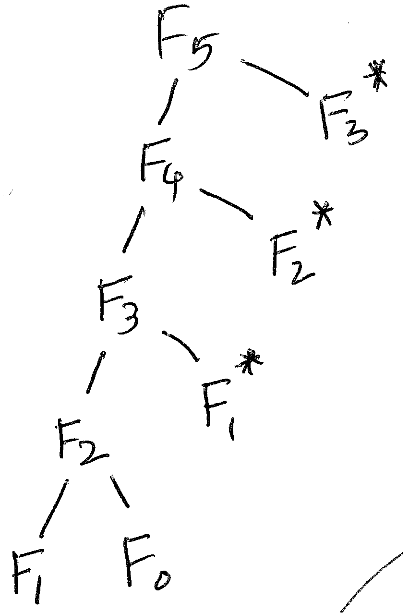
- Only $n+1$ distinct recursive calls

  $F(0), F(1), \ldots, F(n)$.

- Inefficieny is due to big redunduncy, e.g. after computing F_3, at $(*)$, there was no need to recompute it at $(**)$. One could have stored the solution.

Modified Algorithm

- Store solution to every "subproblem" that has been solved.



Store $F_1 = 1$

Store $F_0 = 0$

store $F_2 = 1$

store $F_3 = 2$

store $F_4 = 3$

stored values of F_1, F_2, F_3 used at (*).

This is really the same as <u>iterative algorithm</u> that simply computes

$$F_0, F_1, F_2, \ldots, F_n$$

sequentially in that order.

Dynamic Programming

- Identify a set of polynomially many subproblems such that the original problem is included as one of these subproblems.

- Identify an order from smallest to the largest subproblem,

- Identify a recurrence formula that allows one to compute a subproblem given solutions to all the smaller subproblems.

Fibonacci Example:

Subproblems. $F(0), F(1), \ldots, F(n)$.

order. $\longrightarrow$

Recurrence $F_i = F_{i-1} + F_{i-2}$.

- Dynamic Prog. Algorithm then simply computes solutions to all the subproblems in the given order using the recursive formula.

Subset Sum (with bounded integers)

Problem Given positive integers $a_1, a_2, \dots, a_n$ such that $\forall i,\ a_i \in \{1, 2, 3, \dots, W\}$,

$$W = n^2.$$

Given a number b (s.t. $1 \le b \le n \cdot W$).

Goal is to decide whether there is a subset $S \subseteq \{1, 2, \dots, n\}$ s.t.

$$\sum_{i \in S} a_i = b.$$

Note: Algorithms usually solve the decision problem and also find an actual solution. (e.g. the desired set S above if one exists).

Note: Subproblems usually correspond to prefixes, suffixes, or consecutive subsequences of some given sequence.

\# prefixes, \# suffixes $= O(n)$.
\# consecutive subsequences $= O(n^2)$.

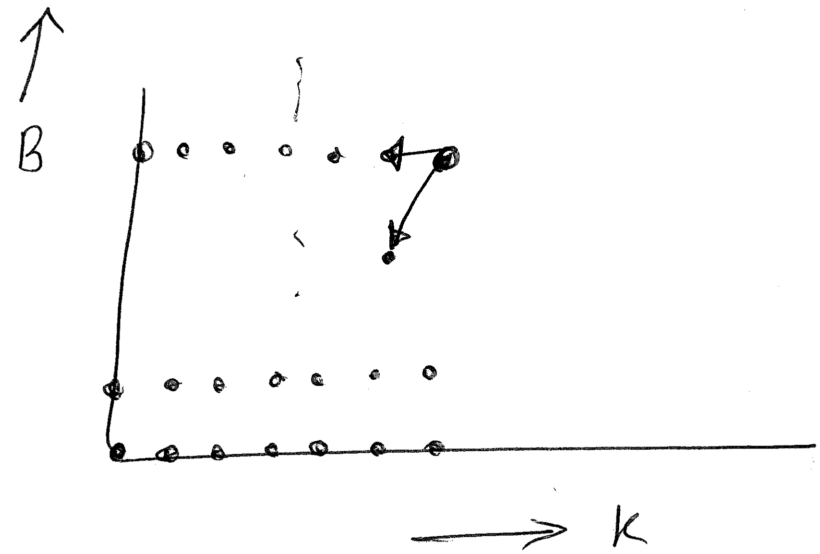
\# subproblems must be at most polynomial in n.

Either there is a subset of $\{a_1, \dots, a_K\}$ that sums to B

Or there is a subset of $\{a_1, \dots, a_K\}$ that sums to $B - a_{K+1}$.

In the latter case, if $T \subseteq \{a_1, \dots, a_K\}$ sums to $B - a_{K+1}$, then $S = T \cup \{K+1\}$ sums to B.

<u>Order</u>.



Can be ordered as per increasing value of k <u>and</u> for given k, as per increasing order of B.

Subproblems

$W = n^2$

There is a subproblem

$$\text{SUM}(\{a_1, \dots, a_k\}, B)$$

for every $1 \le k \le n$ and every $0 \le B \le n \cdot W$.

- # subproblems is at most $n \cdot (nW) = n^4$.
- original problem is $\text{SUM}(\{a_1, \dots, a_n\}, b)$.

The intention is that

$$\text{SUM}(\{a_1, \dots, a_k\}, B) = \begin{cases} \text{YES} & \text{if } \exists S \subseteq \{1, \dots, k\} \text{ s.t. } \sum_{i \in S} a_i = B \\ \text{NO} & \text{otherwise.} \end{cases}$$

Recursive formula

$$\text{SUM}(\{a_1, \dots, a_{k+1}\}, B) = \text{SUM}(\{a_1, \dots, a_k\}, B)$$
$$\vee \ \text{SUM}(\{a_1, \dots, a_k\}, B - a_{k+1}).$$

logical OR

In words, there is a subset of $\{a_1, \dots, a_{k+1}\}$ that sums to B <u>iff</u>

Initialization

$$SUM(\{a_1\}, B) = \begin{cases} YES & \text{if } a_1 = B \\ NO & \text{otherwise} \end{cases}$$

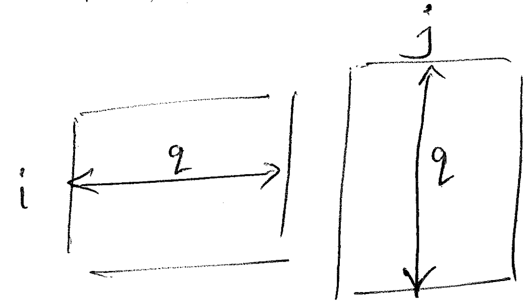
$$SUM(\{a_1, \ldots, a_k\}, 0) = YES \quad (\because S \subseteq \{1, \ldots k\} \text{ can be taken as empty})$$

Matrix Chain Multiplication

Def Given two matrices A, B of sizes $p \times q$, $q \times r$ respectively

$C = A \cdot B$ is a $p \times r$ matrix s.t.

$$C_{ij} = \sum_{k=1}^{q} A_{ik} \cdot B_{kj} \quad \text{for } \begin{matrix} 1 \le i \le p \\ 1 \le j \le r. \end{matrix}$$



$A \cdot B = C.$

Time to compute $= p \cdot q \cdot r.$

Fact Given matrices $\underset{p\times q}{A}, \underset{q\times r}{B}, \underset{r\times s}{C}$,

$(A\cdot B)\cdot C = A\cdot (B\cdot C)$. Associativity,

However time required in these two cases could be very different.

Example $\underset{1\times q}{A} \quad \underset{q\times 1}{B} \quad \underset{1\times q}{C}$.

Time to compute $(A\cdot B)\cdot C = 1\cdot q\cdot 1 + 1\cdot 1\cdot q = 2q$

" $A\cdot(B\cdot C) = q\cdot 1\cdot q + 1\cdot q\cdot q = 2q^2$.

Problem Given n matrices $A_1, A_2, \ldots, A_n$

s.t. A_i has size $p_i \times p_{i+1}$

find the minimum cost way to compute the multiplication

$A_1\cdot A_2\cdot \cdots \cdot A_n$.

Note. **Example** $n=4$. There are 5 possible ways.

$((1\,2)(3\,4))$, $(((1,2),3),4)$, $(1(2(3,4)))$,

$((1(2\,3))4)$ $(1((2\,3)4))$

The # ways C_n is called "Catalan number".

$C_n = \frac{1}{2n-1}\binom{2n-1}{n-1}$. C_n is exponential in n.

Dynamic Programming based algorithm

<u>Idea</u> Suppose the "outermost" product is

$$A_1 \cdot A_2 \cdots A_n = (A_1 \cdot A_2 \cdots A_k) \cdot (A_{k+1} \cdots A_n).$$

If cost( ) denotes optimal cost of computing product then

$$\text{cost}(A_1 \cdot A_2 \cdots A_n) = \text{cost}(A_1 \cdot A_2 \cdots A_k) + \text{cost}(A_{k+1} \cdots A_n) + P_1 \cdot P_{k+1} \cdot P_n.$$

We can try out all "splitting points" k and take the best.

Subproblems

For every $1 \le i \le j \le n$, there is a subproblem

$$\text{cost}(A_i \cdot A_{i+1} \cdots A_j)$$

~~thast~~ that asks for the minimum cost of computing the product $A_i A_{i+1} \cdots A_j$.

subproblems is $O(n^2)$.

Original problem is $A_1 A_2 \cdots A_n$ i.e. $i = 1$, $j = n$.

Recurrence • For $i < j$,

$$cost(A_i A_{i+1} \cdots A_j) = \min_{i \le k < j} \left\{ cost(A_i \cdots A_k) + cost(A_{k+1} \cdots A_j) + P_i P_{k+1} P_j \right\}.$$

• For $i = j$, $cost(A_i) = 0$.

Order According to increasing order of length of the consecutive subsequence $A_i A_{i+1} \cdots A_j$.

— x —

Longest Common Subsequence

Example

$X = A\,G\,C\,G\,T\,A\,G$

$Y = G\,T\,C\,A\,G\,A$

Def $Z = (z_1 \cdots z_k)$ is a subsequence of $X = (x_1 x_2 \cdots x_n)$ if there are indices $1 \le i_1 < i_2 \cdots < i_k \le n$ such that $z_1 = x_{i_1}$, $z_2 = x_{i_2}$, $\ldots$, $z_k = x_{i_k}$.

Problem Given two sequences $X = (x_1 x_2 \cdots x_n)$
$Y = (y_1 y_2 \cdots y_m)$,

find L.C.S. i.e. sequence Z that is a subsequence of both X and Y and s.t. $|Z|$ is maximum.

Idea Given $X = x_1 x_2 \cdots x_n$
$Y = y_1 y_2 \cdots y_m$,

one can begin by considering cases depending on whether x_1 is "matched" with y_1.

Subproblems - All pairs of suffixes
$(x_i x_{i+1} \cdots x_n, \; y_j y_{j+1} \cdots y_m)$.

- \# subproblems $= O(m \cdot n)$.
- Original problem corresponds to $i=1$, $j=1$.

Let
$LCS(x_i x_{i+1} \cdots x_n, \; y_j y_{j+1} \cdots y_m)$ denote the length of the largest common subseq.

Recursive formula

$$LCS(x_i x_{i+1} \cdots x_n, y_j y_{j+1} \cdots y_m)$$

$$= \max \begin{cases} LCS(x_{i+1} \cdots x_n, \; y_j y_{j+1} \cdots y_m) \\ LCS(x_i x_{i+1} \cdots x_n, \; y_{j+1} \cdots y_m) \\ 1 + LCS(x_{i+1} \cdots x_n, \; y_{j+1} \cdots y_m) \quad \underline{\text{if } x_i = y_j}. \end{cases}$$

Order In increasing order of sum of lengths of two sequences i.e. $|n-i+1| + |m-j+1|$.

Base case

$LCS(x_n, \phi) = 0$ $\quad LCS(\phi, y_m) = 0,$

$LCS(\phi, \phi) = 0.$

Note. There are three choices in $\max \begin{cases} \#1 \\ \#2 \\ \#3 \end{cases}$.

While computation, one can keep track of which of the three choices was the best (maximum) one. This information can be used to <u>find the LCS</u> and <u>not just the length of the LCS</u>.