Dynamic Programming

➯ Objective:

- Dynamic programming is applied to optimization problems.

➯Comparison

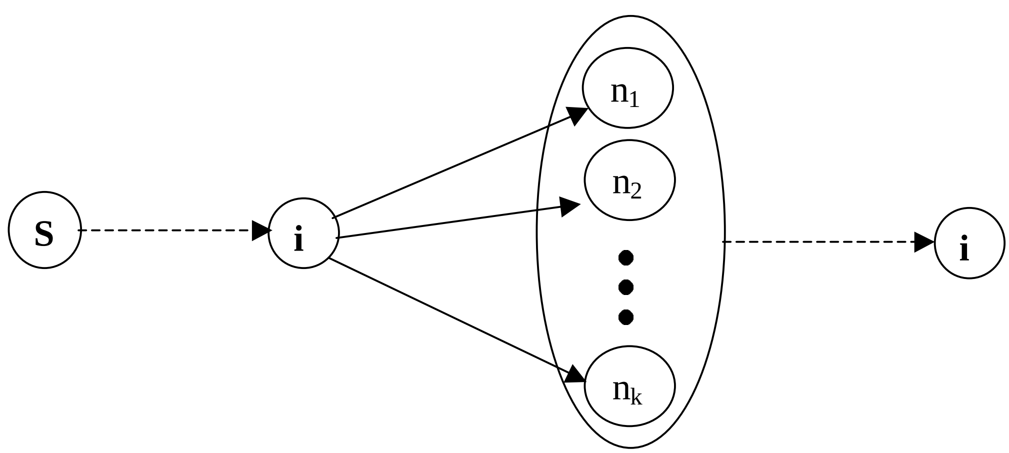
- Divide-and-conquer algorithms partition the problem into independent sub problems.

- Greedy method generates a single decision "locally optimal", at each time.

➯example:

- S P: form S→d

- At any given node i:



Q: Which of the n_i will be chosen in the S.P. from s to d. ?

Note: There is no way to make the right choice or decision at this time & guarantee that future decisions lead to Optimal Solution.

- Principle of optimality
    - An optimal sequence of decisions has the property that what ever the initial state and decision are, the remaining decisions must constitute an optimal decision sequence with regard to the state resulting from the first decision.

- Principle:

    - A sub solution for an optimal solution is an optimal solution for the sub problem.

- Dynamic Programming:

    - Uses the principle of optimality.

- Example:

    - All pairs shortest paths.
    - Matrix- chain.
    - Optimal binary search tree.

- General approach:

    - Characterize the structure of an optimal solution.
    - Recursively define the value of an optimal solution.
    - Compute the value of an optimal solution in bottom-up fashion.

Matrix-chain multiplication

➱ Requirements:

- Input: A sequence (chain) $A_1, A_2, \ldots, A_n$ of n matrices where A_i has dimension p_{i-1} x p_i for $1 \le i \le n$
- Output: The product $A_1\ A_2\ A_3 \ldots A_n$ such that the total number of scalar multiplications is minimized.

➱ Example

Given these three matrices: $A_1=(4,2)$, $A_2=(2,3)$, $A_3=(3,4)$, let us compute $A_1 * A_2 * A_3$

* S1 = $A_1 * A_2 * A_3 = (A_1 * A_2) * A_3$

  B= $A_1 * A_2$ takes 4x2x3=24 and B=(4,3)

  B*A_3 takes 4x3x4=48

  ➔ Cost $((A_1 * A_2) * A_3)=24+48=72$

* S2 = $A_1 * A_2 * A_3 = A_1 * (A_2 * A_3)$

  C= $A_2 * A_3$ takes 2x3x4=24 and C(2,4)

  A_1*C takes 4x2x4=32

  ➔ Cost $(A_1 * (A_2 * A_3))=24+32=56$

* Compare S1 and S2?

- Characterization of the structure of an optimal solution:

  - An optimal solution of the product $A_1 A_2 \ldots A_n$ splits the product between A_k and A_{k+1} for some integer k: $1 \le k < n$

$$\underbrace{A_1 A_2 \ldots A_k}_{C_1} \quad \text{and} \quad \underbrace{A_{k+1} \ldots A_n}_{C_2}$$

such that

(A) Cost $(A_1 \ldots A_n)$ = Cost $(A_1 \ldots A_k)$ // Optimal
+ Cost $(A_{k+1} \ldots A_n)$ //Optimal
+ Cost $(C_1 C_2)$.

➔ makes Cost $(A_1 \ldots A_n)$ an optimal solution.

- Recursive solution

  - Let m[i,j] be the minimum number of multiplications needed to compute the product $A_i, A_{i+1}, \ldots, A_j$

  - From (A), we have

$$m[i,j]=m[i,k]+m[k+1,j]+p_{i-1}p_k p_j$$

This is if we know the value of k????
➔ k can be any value $I \le k < j$

  - To get the optimal value of m[i,j] ➔ Compute all m[i,j] for all $i \le k < j$ and find the minimum.

$$m[i,j] = \begin{cases} 0 & \text{if } i = j \\ \min\limits_{i \le k < j} \{m[i,k] + m[k+1,j] + p_{i-1} p_k p_j\} & \text{if } i \ne j \end{cases}$$

Algorithm:

```
Procedure matrix_chain
Begin
        For i=1 to n do
                M[i,i]=0;
        End;
        For L=2 to n do   /* length of chains: 2-matrices, 3-matrices, etc. */
                For i =1 to n-L+1 do    /*n-L+1: position of last chain */
                        j=i+l-1;
                        M[i,j] = ∞;
                        For k=i to j-1 do
                                Q = m[i,k]+m[k+1,j]+p_{i-1}p_k p_j;
                                If q < m[i,j]
                                Then
                                        M[i,j] = q;
                                End if;
                        End for;
                End for;
        End for;
End;
```

Complexity: $O(n^3)$.

All pairs shortest paths

- Problem:

  ✓ Input: G=(V,E) is a directed graph
  A(1..n,1..n) is the cost matrix of G

$$C(i,j)=\begin{cases} 0 & i=j \\ \infty & \langle i,j\rangle \notin E \\ wij & wij \text{ is the weight of } \langle i,j\rangle \text{ if } \langle i,j\rangle \in E \end{cases}$$

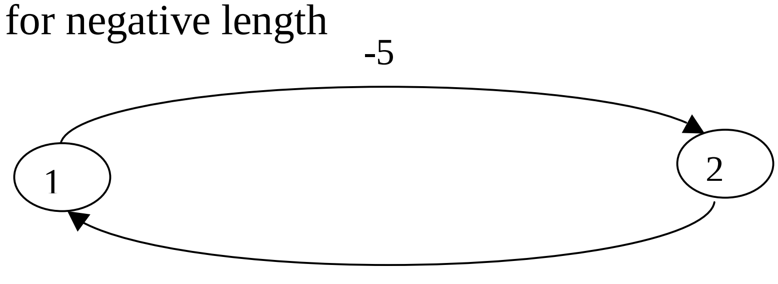
  ✓ Output: matrix A(1..n,1..n) such that A(i,j)= the shortest path from i to j where 1<=I,j<=n

- First method:

  - Apply the single shortest path algorithm for each vertex of V
  - Complexity: $O(n^3)$.

- Second method: Floyd-warshall algorithm

  - Not all C(I,j) >=0;
  - G has no cycle for negative length



2

Shortest path between 1 and 2: 1, 2, 1, 2, 1, 2, 1, 2 ...
$-1 \quad -1$

➭ Intermediate vertex:

- Definition: An intermediate vertex of a simple path $p=(v_1,v_2,\ldots v_{j-1},v_j)$ is any vertex of p other than $v_1$or v_j, that is any vertex in the set $\{v_2,\ldots.v_{j-1}\}$.

➭ Characterization of the structure of an optimal solution:

- p = (i,……..,j) for every i and every j in G. The intermediate vertices of p are in {1,2,….,k-1}.

- Consider the next intermediate vertex k:

- If k is an intermediate vertex, otherwise it is not considered.

S.t. p_1 = (i,……,k) & p_2 = (k,……,j) have their intermediate vertices in {1,2,…..,k-1}.

$p = p_1, p_2$.

➭ Recursive solution:

$A^k(i,j)= \min (A^{k-1}(i,j), A^{k-1}(i,k)+A^{k-1}(k,j))$ For $k \geq 1$.

∗ $A^{k-1}(i,j)$ is the shortest path including only the intermediate vertices in {1,2,….,k-1}.
∗ $A^k(i,j)$ is the shortest path including only the intermediate vertices in {1,2,….,k}.

- Algorithm:

```
Floy-warshall(cost(1:n),A(1:n)).
integer i,j,k ;
Begin
    for k =1 to n do.
        for I = 1 to n do.
            for j =1 to n do.
                A(i,j) = min(A(i,j),A(i,k)+A(k,j))
            end for
        end for
    end for
end.
```

Complexity: $O(n^3)$.