Course Name: Analysis and Design of Algorithms

Topics to be covered

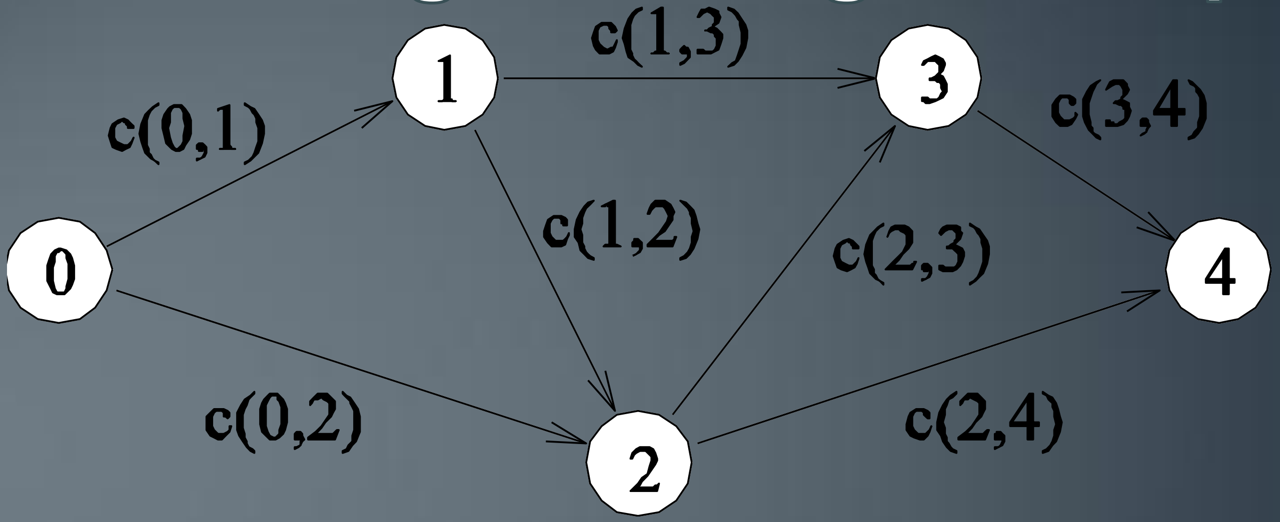
- Dynamic Programming
  - 0/1 Knapsack Problem

Dynamic Programming: Example

- Consider the problem of finding a shortest path between a pair of vertices in an acyclic graph.
- An edge connecting node *i* to node *j* has cost *c(i,j)*.
- The graph contains *n* nodes numbered *0,1,…, n-1*, and has an edge from node *i* to node *j* only if *i < j*. Node 0 is source and node *n-1* is the destination.
- Let *f(x)* be the cost of the shortest path from node 0 to node *x*.

$$f(x) = \begin{cases} 0 & x = 0 \\ \min\limits_{0 \leq j < x} \{f(j) + c(j,x)\} & 1 \leq x \leq n-1 \end{cases}$$

Dynamic Programming: Example



- A graph for which the shortest path between nodes 0 and 4 is to be computed.

$$f(4) = \min\{f(3) + c(3,4), f(2) + c(2,4)\}.$$

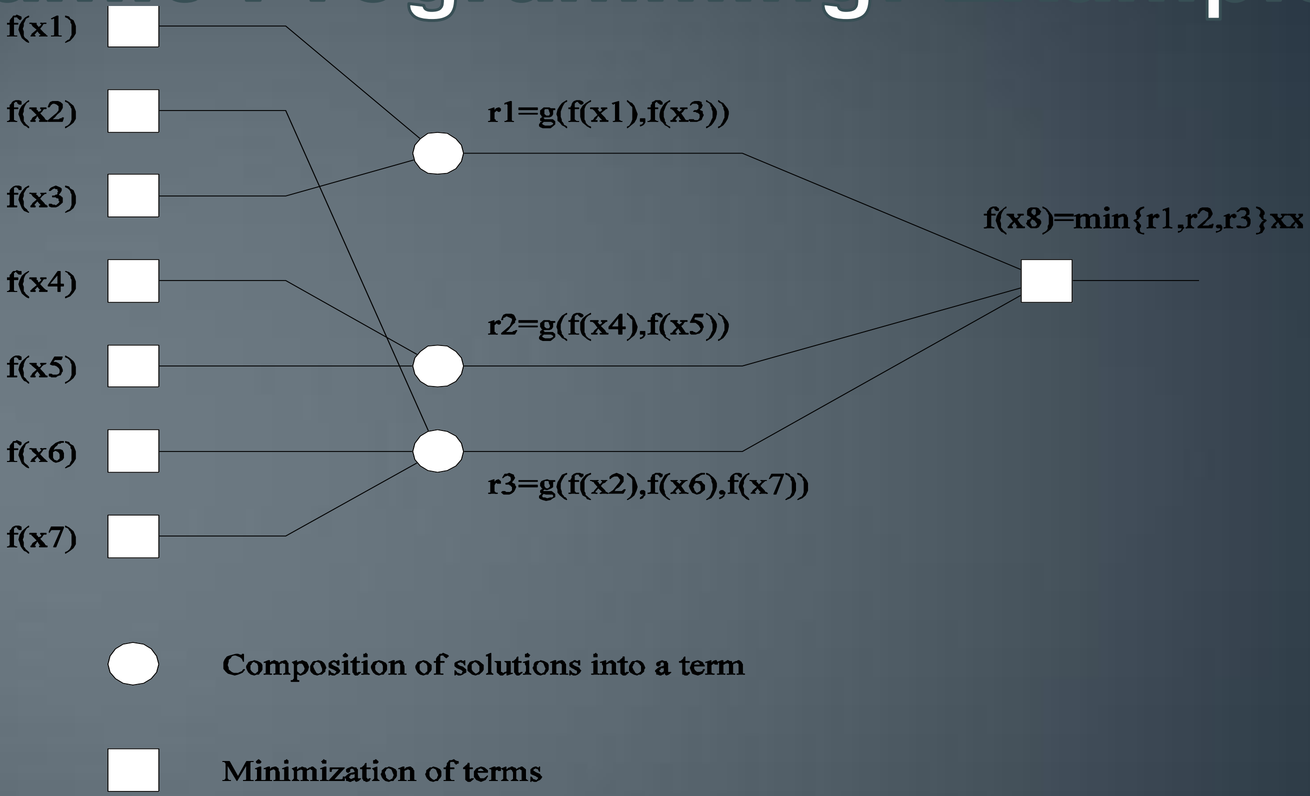
Dynamic Programming

- The solution to a DP problem is typically expressed as a minimum (or maximum) of possible alternate solutions.
- If *r* represents the cost of a solution composed of subproblems x_1, x_2, …, x_l, then *r* can be written as

  $$r = g(f(x_1), f(x_2), \ldots, f(x_l)).$$

  Here, *g* is the *composition function*.
- If the optimal solution to each problem is determined by composing optimal solutions to the subproblems and selecting the minimum (or maximum), the formulation is said to be a DP formulation.

Dynamic Programming: Example



The computation and composition of subproblem solutions to solve problem $f(x_8)$.

Dynamic Programming

- The recursive DP equation is also called the *functional equation* or *optimization equation*.
- In the equation for the shortest path problem the composition function is $f(j) + c(j,x)$. This contains a single recursive term ($f(j)$). Such a formulation is called monadic.
- If the RHS has multiple recursive terms, the DP formulation is called polyadic.

Dynamic Programming

- The dependencies between subproblems can be expressed as a graph.
- If the graph can be levelized (i.e., solutions to problems at a level depend only on solutions to problems at the previous level), the formulation is called serial, else it is called non-serial.
- Based on these two criteria, we can classify DP formulations into four categories - serial-monadic, serial-polyadic, non-serial-monadic, non-serial-polyadic.
- This classification is useful since it identifies concurrency and dependencies that guide parallel formulations.

0/1 Knapsack Problem

- We are given a knapsack of capacity c and a set of n objects numbered $1,2,\dots,n$. Each object i has weight w_i and profit p_i.
- Let $v = [v_1, v_2, \dots, v_n]$ be a solution vector in which $v_i = 0$ if object i is not in the knapsack, and $v_i = 1$ if it is in the knapsack.
- The goal is to find a subset of objects to put into the knapsack so that

$$\sum_{i=1}^{n} w_i v_i \leq c$$

(that is, the objects fit into the knapsack) and

$$\sum_{i=1}^{n} p_i v_i$$

is maximized (that is, the profit is maximized).

0/1 Knapsack Problem

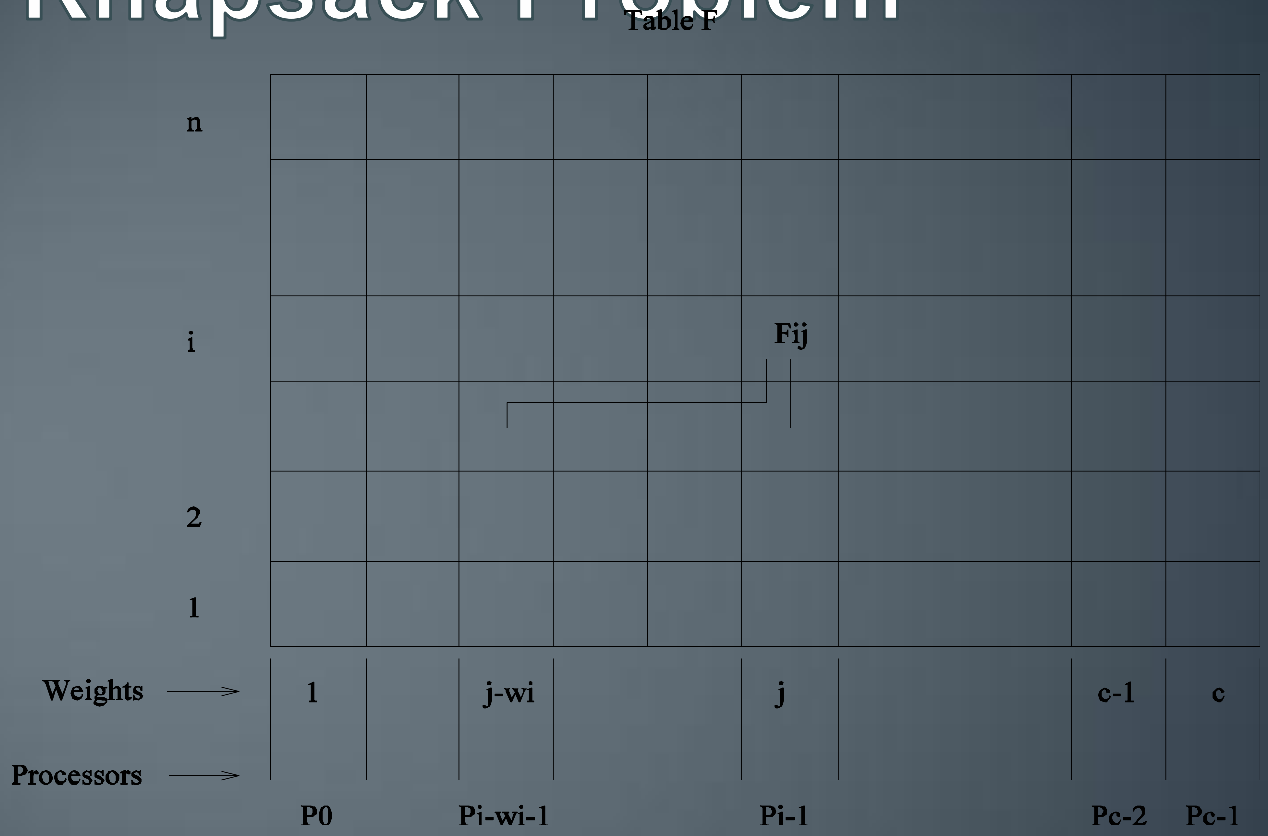
- The naive method is to consider all *2^{n}* possible subsets of the *n* objects and choose the one that fits into the knapsack and maximizes the profit.
- Let *F[i,x]* be the maximum profit for a knapsack of capacity *x* using only objects *{1,2,…,i}*. The DP formulation is:

$$F[i,x] = \begin{cases} 0 & x \geq 0, i = 0 \\ -\infty & x < 0, i = 0 \\ \max\{F[i-1,x], (F[i-1,x-w_i] + p_i)\} & 1 \leq i \leq n \end{cases}$$

0/1 Knapsack Problem

- Construct a table *F* of size *n x c* in row-major order.
- Filling an entry in a row requires two entries from the previous row: one from the same column and one from the column offset by the weight of the object corresponding to the row.
- Computing each entry takes constant time; the sequential run time of this algorithm is $\Theta(nc)$.
- The formulation is serial-monadic.

0/1 Knapsack Problem



Computing entries of table *F* for the 0/1 knapsack problem. The computation of entry *F[i,j]* requires communication with processing elements containing entries *F[i-1,j]* and $F[i-1,j-w_i]$.

0/1 Knapsack Problem

- Using c processors in a PRAM, we can derive a simple parallel algorithm that runs in $O(n)$ time by partitioning the columns across processors.
- In a distributed memory machine, in the j^{th} iteration, for computing $F[j,r]$ at processing element P_{r-1}, $F[j-1,r]$ is available locally but $F[j-1,r-w_j]$ must be fetched.
- The communication operation is a circular shift and the time is given by $(t_s + t_w) \log c$. The total time is therefore $t_c + (t_s + t_w) \log c$.
- Across all n iterations (rows), the parallel time is $O(n \log c)$. Note that this is not cost optimal.