IE 400: Principles of Engineering Management

Set 9: Dynamic Programming

Dynamic Programming

- Transforms a complex optimization problem into a sequence of smaller ones (divide and conquer)
- Problem is divided into a sequence of stages
- Each stage has associated states
- Relies on recursion and principle of optimality
- Forward or backward pass to optimize
- Most general of all solution methodologies

Example: Resource Allocation Problem

- A company has \$6000 to invest and 3 investments are available. If d_j thousands of dollars are invested in investment j, then a net present value (in thousands) of $r_j(d_j)$ is obtained.
- In particular,

$$r_1(d_1) = \begin{cases} 7d_1 + 2 & d_1 > 0 \\ 0 & d_1 = 0 \end{cases}$$

$$r_2(d_2) = \begin{cases} 3d_2 + 7 & d_2 > 0 \\ 0 & d_2 = 0 \end{cases}$$

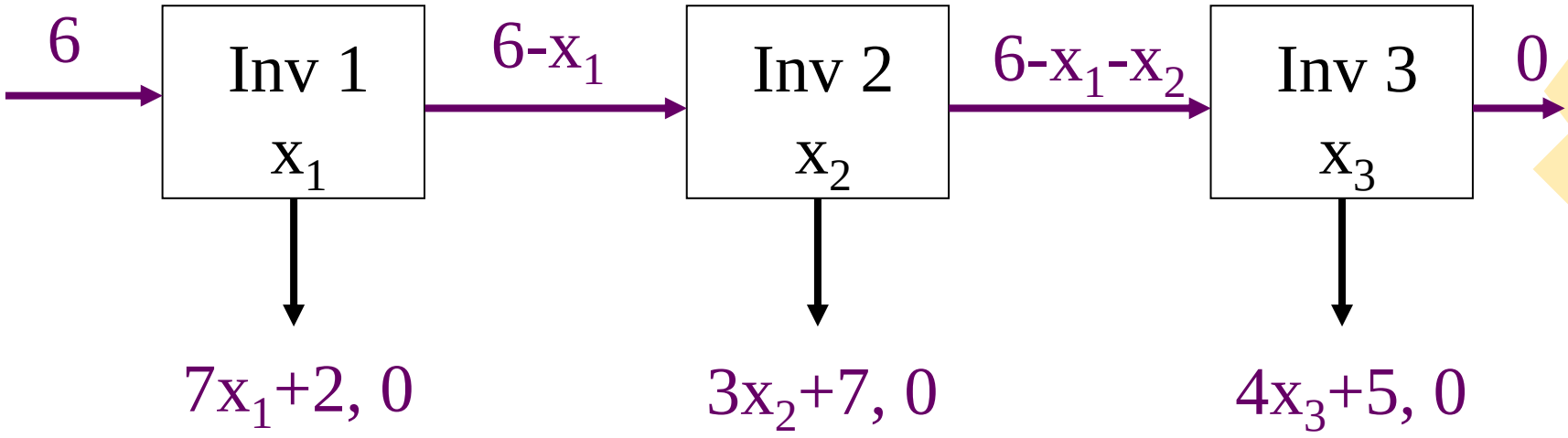
$$r_3(d_3) = \begin{cases} 4d_3 + 5 & d_3 > 0 \\ 0 & d_3 = 0 \end{cases}$$

Example – cont'd.

- The amount placed in each investment must be an integer multiple of $1000.
- How should the company allocate $6000 to maximize the net present value?

$$\begin{aligned} \max \quad & r_1(x_1) + r_2(x_2) + r_3(x_3) \\ s.t. \quad & x_1 + x_2 + x_3 = 6 \\ & x_1, x_2, x_3 \geq 0 \quad \text{and integer} \end{aligned}$$

DP Functions



In general, n such investments:

- $f_k(d)$ = max net present value by investing d dollars in investments k, k+1, k+2, … , n

- $f_1(6)$: optimal value
  compute using f_3, f_2, and f_1 values

DP Recursion, Principle of Optimality, and Boundary Conditions

$$f_k(d) = \text{maximize} \quad \{r_k(x_k) + f_{k+1}(d - x_k)\} \quad \text{for k} \leq \text{n-1}$$

$$0 \leq x_k \leq d :$$

$$x_k \quad \text{integer}$$

- Principle of optimality: optimal decision at stage k should use optimal decision of later stages

$$f_n(d) = r_n(d) \qquad \text{for d} \geq 0$$

- To trace the best solution:
- Let $x_k(d)$ = amount to invest in alternative k if d dollars are available for alternatives k, k+1, … , n

Solving with DP

1) Identify DP Function

$f_k(d)$ = max net present value by investing d dollars in investments k, k+1, k+2, … , n

2) Construct Recursion

$$f_k(d) = \text{maximize} \quad \{r_k(x_k) + f_{k+1}(d - x_k)\} \quad \text{for } k \leq n\text{-}1$$

$$0 \leq x_k \leq d :$$

$$x_k \ \text{integer}$$

3) Keep track of best solution

$x_k(d)$ = amount to invest in alternative k if d dollars are available for alternatives k, k+1, … , n

Solving with DP

4) Identify the base case (initiate recursion)

$$f_n(d) = r_n(d) \qquad \text{for d} \geq 0$$

5) Identify optimal value

- $f_1(6)$: optimal value

6) Starting with the base case solve for all necessary function values

Solutions to Recursive Functions (Stage 3)

$$f_3(0) = 0 \qquad x_3(0) = 0$$
$$f_3(1) = 9 \qquad x_3(1) = 1$$
$$f_3(2) = 13 \qquad x_3(2) = 2$$
$$f_3(3) = 17 \qquad x_3(3) = 3$$
$$f_3(4) = 21 \qquad x_3(4) = 4$$
$$f_3(5) = 25 \qquad x_3(5) = 5$$
$$f_3(6) = 29 \qquad x_3(6) = 6$$

Solutions to Recursive Functions (Stage 2)

$$f_2(d) = \text{maximize} \quad \{r_2(x_2) + f_3(d - x_2)\}$$

$$0 \le x_2 \le d :$$

$$x_2 \text{ integer}$$

$$x_2(d) = \text{best value above}$$

$$f_2(0) = 0 \qquad x_2(0) = 0$$

$$f_2(1) = 10 \qquad x_2(1) = 1$$

$$f_2(2) = 19 \qquad x_2(2) = 1$$

$$f_2(3) = 23 \qquad x_2(3) = 1$$

$$f_2(4) = 27 \qquad x_2(4) = 1$$

$$f_2(5) = 31 \qquad x_2(5) = 1$$

$$f_2(6) = 35 \qquad x_2(6) = 1$$

Solutions to Recursive Functions (Stage 3)

$$f_1(d) = \text{maximize} \quad \{r_1(x_1) + f_2(d - x_1)\}$$

$$0 \le x_1 \le d :$$

$$x_1 \text{ integer}$$

$$x_1(d) = \text{best value above}$$

$$f_1(6) = \max\{35,40,43,46,49,47,44\} = 49$$

$$x_1(6) = 4$$

Trace back the solution: $x_1 = 4, \quad x_2 = 1, \quad x_3 = 1$

$(f_2(2)) \quad (f_3(1))$

Optimal Solution: Invest \$4000 in 1, \$1000 in 2, and \$1000 in 3

Net Present Value = \$49,000

Binary Knapsack Problem

$$\max \quad \sum_{j=1}^{n} c_j x_j$$

$$s.t. \quad \sum_{j=1}^{n} a_j x_j \leq b$$

$$x_j \in \{0,1\} \text{ for } j=1, \ldots, n$$

DP Function:

$$f_k(d) = \max \quad \sum_{j=1}^{k} c_j x_j \qquad \text{for } d=0,1,\ldots b$$

$$s.t. \quad \sum_{j=1}^{k} a_j x_j \leq d \qquad k=1,2,\ldots n$$

$$x_j \in \{0,1\} \text{ for } j=1, \ldots, k$$

Optimal Value: $f_n(b)$

Binary Knapsack Problem

Boundary Conditions: $$f_1(d) = \begin{cases} c_1 & \text{if} \quad a_1 \le d \\ 0 & \text{otherwise} \end{cases}$$

Recursion using principle of optimality: *for* $k=2,\ldots n$

$b=0,\ldots,b$

$$f_k(d) = \begin{cases} f_{k-1}(d) & \text{if} \quad a_k > d \\ \max\{ f_{k-1}(d) \ , \ c_k + f_{k-1}(d - a_k) \ \} & \text{if} \quad a_k \le d \\ \quad x_k = 0 \qquad\qquad x_k = 1 & \end{cases}$$

To trace: Let $x_k(d)$ be the best value of this variable

Example: Binary Knapsack Problem

$$\max \quad 16x_1 + 19x_2 + 23x_3 + 28x_4$$

$$s.t. \quad 2x_1 + 3x_2 + 4x_3 + 5x_4 \leq 7$$

$$x_i \in \{0,1\} \quad \text{for} \quad i=1,...,4$$

$$f_1(d) = \begin{cases} 0 & \text{if} \quad d=0,1 \quad x_1 = 0 \\ 16 & \text{if} \quad d \geq 2 \quad x_1 = 1 \end{cases}$$

$$f_2(d) = \begin{cases} 0 & \text{if} \quad d=0,1 \quad x_2 = 0 \\ 16 & \text{if} \quad d = 2 \quad x_2 = 0 \\ 19 & \text{if} \quad d = 3,4 \quad x_2 = 1 \\ 35 & \text{if} \quad d = 5,6,7 \quad x_2 = 1 \end{cases}$$

Example: Binary Knapsack Problem

$$\max \quad 16x_1 + 19x_2 + 23x_3 + 28x_4$$

$$s.t. \quad 2x_1 + 3x_2 + 4x_3 + 5x_4 \leq 7$$

$$x_i \in \{0,1\} \quad \text{for} \quad i=1,...,4$$

$$f_3(d) = \begin{cases} f_2(d) & \text{if} \quad d=0,1,2,3 \quad x_3 = 0 \\ 23 & \text{if} \quad d = 4 \quad x_3 = 1 \\ 35 & \text{if} \quad d = 5 \quad x_3 = 1 \\ 39 & \text{if} \quad d = 6 \quad x_3 = 1 \\ 42 & \text{if} \quad d = 7 \quad x_3 = 1 \end{cases}$$

$$f_4(7) = \max(42, 28 + f_3(2)) = 44 \qquad x_4 = 1$$

Hence, $x_4^* = 1, \quad x_3^* = 0, \quad x_2^* = 0, \quad x_1^* = 1$

General Knapsack Problem

$$\max \quad \sum_{j=1}^{n} c_j x_j$$

$$s.t. \quad \sum_{j=1}^{n} a_j x_j \leq b$$

$$x_j \geq 0 \quad \text{and integer for j=1, ..., n}$$

- **DP Function:**

$$g(w) = \max \quad \sum_{j=1}^{n} c_j x_j$$

$$s.t. \quad \sum_{j=1}^{n} a_j x_j \leq w$$

$$x_j \geq 0 \quad \text{and integer for j=1, ..., n}$$

- **Optimal Value:** g(b)

General Knapsack Problem

- **Boundary Conditions:**

$$g(w) = 0 \qquad \text{for} \qquad 0 \le w < \min_i \{a_i\}$$

- **Recursion using principle of optimality**

$$g(w) = \max_{j:\, a_j \le w} \{c_j + g\,(w - a_j)\} \qquad \text{for } w \ge \min_j \{a_j\}$$

- **To trace:** Let x (w) be the index of the variable chosen

General Knapsack Problem

$$\begin{aligned} \max \quad & 11x_1 + 7x_2 + 12x_3 \\ s.t. \quad & 4x_1 + 3x_2 + 5x_3 \le 10 \\ & x_i \in \{0,1\} \quad \text{for} \quad i=1,...,3 \end{aligned}$$

$g(0) = g(1) = g(2) = 0$

$g(3) = 7$	$x(3) = 2$
$g(4) = 11$	$x(4) = 1$
$g(5) = 12$	$x(5) = 3$
$g(6) = 14$	$x(6) = 2$
$g(7) = 18$	$x(7) = 1$ or 2
$g(8) = 22$	$x(8) = 1$
$g(9) = 23$	$x(9) = 1$ or 3

optimal value $g(10) = 25$,

x(10)=1 or 2

optimal solution:

$$x_1^* = 1, \quad x_2^* = 2, \quad x_3^* = 0$$

Dynamic Programming Approach to Shortest Path Problems

- Shortest path algorithms exploit relationships among optimal paths (and path lengths) for different pairs of nodes

Functional Notation

single source shortest path problem

- v[k] : the length of a shortest path from the source node to node k
- If no path exists, v[k]=∞
- If optimal paths must have optimal subpaths, it follows that a shortest path can be constructed by extending known shortest paths
- Functional equations of a dynamic program encode the recursive connections among optimal solution values that are implied by a principle of optimality
    - In a graph with no negative directed cycles, optimal paths must have optimal subpaths

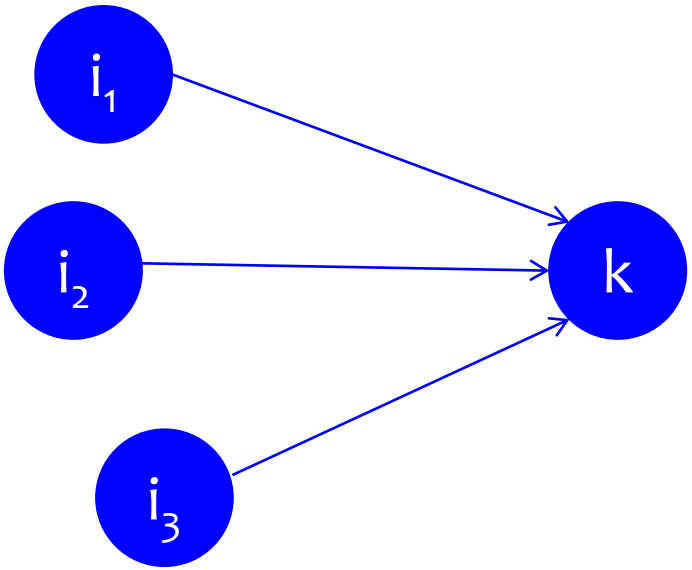
Functional Equations

$v[s]=0$

$v[k]=\min_{i:(i,k)\in A} \{v[i]+c_{ik}\}$ $\quad \forall k \in N\backslash\{s\}$

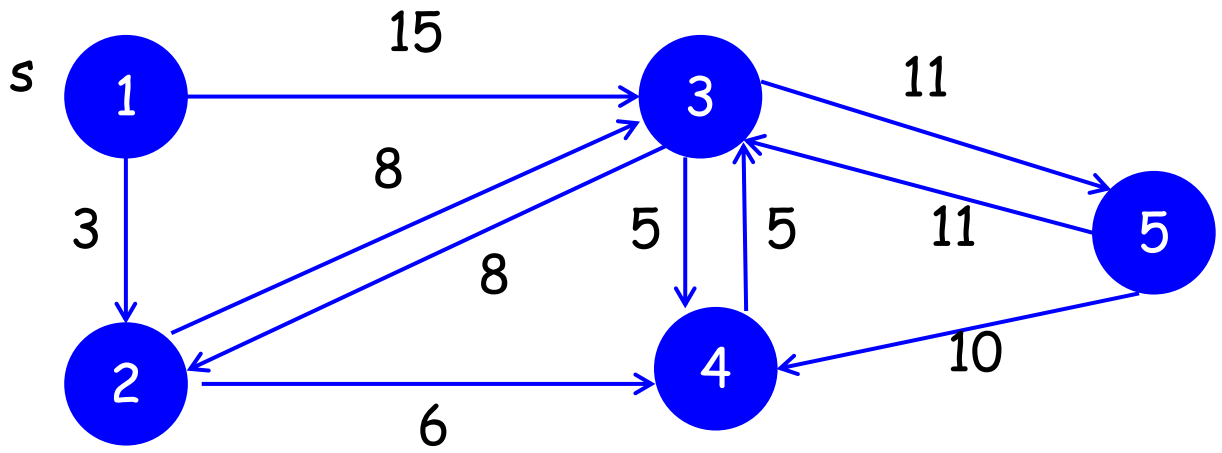
If there is no neighbor i leading to k, then the minimum is ∞

d[k]=preceeding node on the shortest path from s to k



Functional Equations

Example:



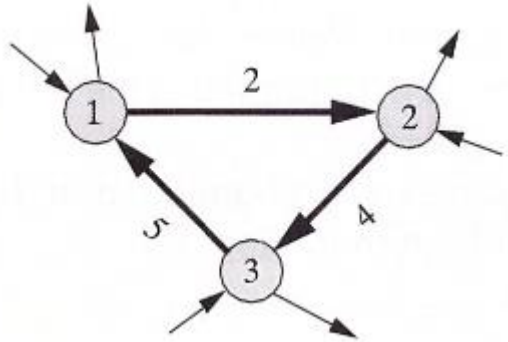
v[1]=0
v[2]=min {v[1]+3,v[3]+8}=3
v[3]=min{v[1]+15,v[2]+8,v[4]+5,v[4]+5,v[5]+11}=11
v[4]=min{v[2]+6,v[3]+5,v[5]+10]}=9
V[5]=∞

Optimal paths must extend optimal subpaths

Functional Equations

How to solve functional equations

- Nonlinear due to min operator
- Because each equation shows an expression for a single value v[k], possible just to evaluate those expressions (one-pass evaluation).
- However, directed cycles introduce circular dependencies in functional equations that preclude their solution by one-pass evaluations



$$v[1] = \min\{v[3] + 5, \ldots\}$$
$$v[2] = \min\{v[1] + 2, \ldots\}$$
$$v[3] = \min\{v[2] + 4, \ldots\}$$

Evaluating the expression for $v[2]$ requires $v[1]$; evaluating the expression for $v[3]$ requires $v[2]$; and evaluating the expression for $v[1]$ requires $v[3]$.

Bellman-Ford Algorithm

Repeated Evaluation Algorithm

- Each major iteration of the search evaluates the functional equations for each v[k] using results from the preceding iteration. The algorithm stops, when the results do not change.
  - $v^{(t)}[k]$: value of v[k] obtained on the t^{th} iteration
- To be able to know the shortest path at the termination of the algorithm, labels d[k] keep notes to allow us to recover the optimal paths
  - d[k] : node preceding k in the best known path from s to k

Bellman-Ford Algorithm

Step 0: Initialization. With s the source node, initialize optimal path lengths

$$v^{(0)}[k] \leftarrow \begin{cases} 0 & \text{if } k = s \\ +\infty & \text{otherwise} \end{cases}$$

and set iteration counter $t \leftarrow 1$.

Step 1: Evaluation. For each k evaluate

$$v^{(t)}[k] \leftarrow \min\{v^{(t-1)}[i] + c_{i,k} : (i, k) \text{ exists}\}$$

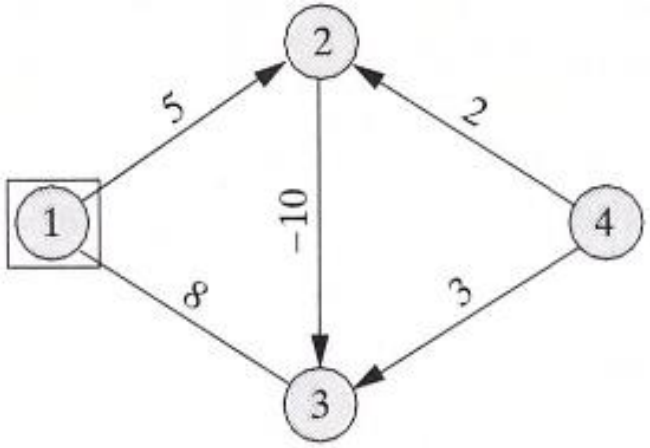
If $v^{(t)}[k] < v^{(t-1)}[k]$, also set $d[k] \leftarrow$ the number of a neighboring node i achieving the minimum $v^{(t)}[k]$.

Step 2: Stopping. Terminate if $v^{(t)}[k] = v^{(t-1)}[k]$ for every k, or if $t =$ the number of nodes in the graph. Values $v^{(t)}[k]$ then equal the required shortest path lengths unless some $v^{(t)}[k]$ changed at the last t, in which case the graph contains a negative dicycle.

Step 3: Advance. If some $v[k]$ changed and $t <$ the number of nodes, increment $t \leftarrow t + 1$ and return to Step 1.

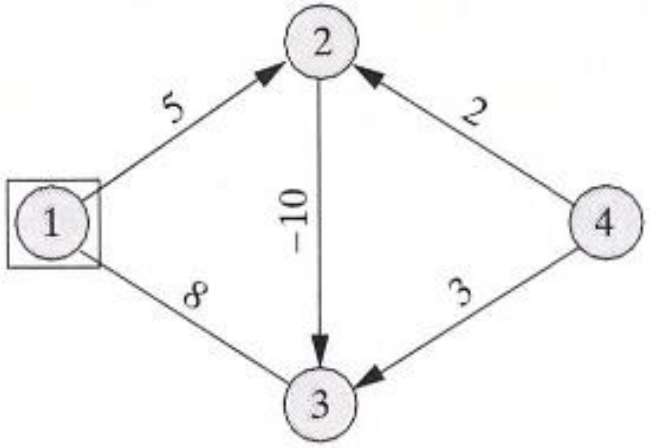
The algorithm works on any graph with no negative directed cycles

Bellman-Ford Algorithm



t	$v^t[1]$	$v^t[2]$	$v^t[3]$	$v^t[4]$		d(1)	d(2)	d(3)	d(4)
0									
1									
2									
3									

Bellman-Ford Algorithm



t	$v^{(t)}[1]$	$v^{(t)}[2]$	$v^{(t)}[3]$	$v^{(t)}[4]$
0	0	$+\infty$	$+\infty$	$+\infty$
1		5	8	
2			−5	
3				

t	$d[1]$	$d[2]$	$d[3]$	$d[4]$
1		1	1	
2			2	
3				

- v[1]=0; v[5]=0; v[3]=-5; v[4]=∞
- A shortest path from source s to any other node k can recovered by starting at k, backtracking to neighboring node d[k], and continuing with an optimal path from s to the neighbor until source is encountered
- Paths: 1-2; 1-2-3

Bellman-Ford Algorithm

Why does the algorithm work?

- Remark that in a graph with n nodes, a path can contain at most n-1 arcs
- At the t^{th} iteration, the length of optimal paths of at most t arcs should be correct
- Moreover, if we have a whole iteration without changing any node's value, no more changes can occur

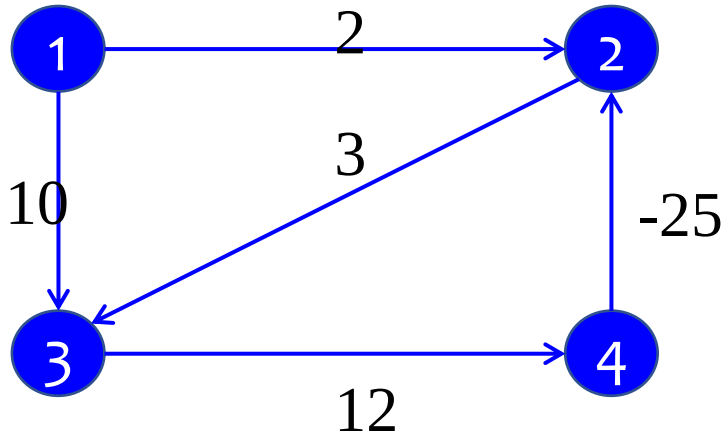
Bellman-Ford Algorithm

Encountering Negative Directed Cycles

If the algorithm encounters negative directed cycles, it demonstrates their presence by continuing to change $v^{(t)}[k]$ on iteration t=the number of nodes. Any node k with such a changing $v^{(t)}[k]$ belongs to a negative directed cycle

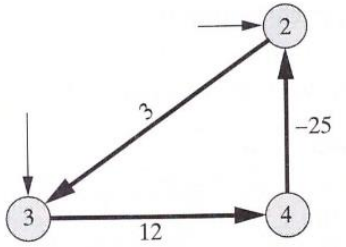
Bellman-Ford Algorithm

Encountering Negative Directed Cycles

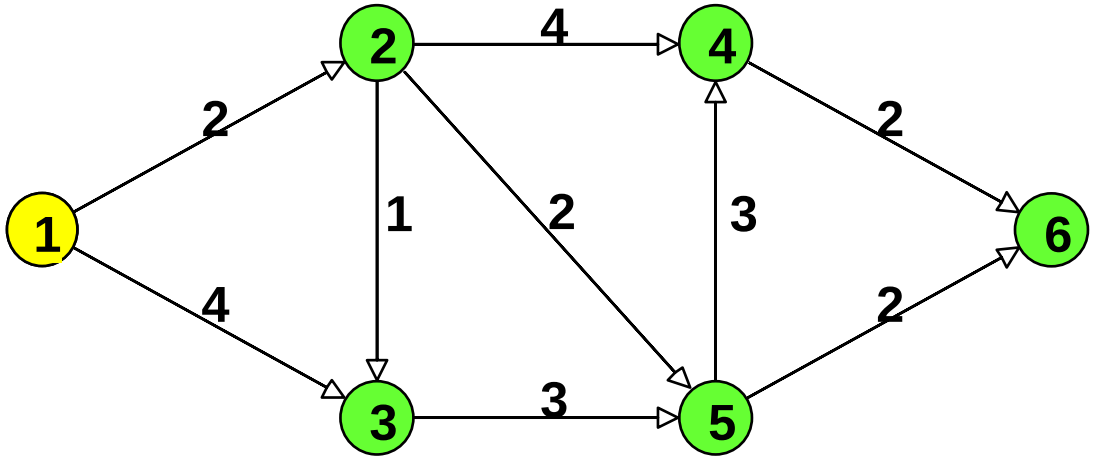


t	$v^{(t)}[1]$	$v^{(t)}[2]$	$v^{(t)}[3]$	$v^{(t)}[4]$	$d[1]$	$d[2]$	$d[3]$	$d[4]$
0	0	$+\infty$	$+\infty$	$+\infty$				
1		2	10			1	1	
2			5	22			2	3
3		–3		17		4		
4		–8	0					

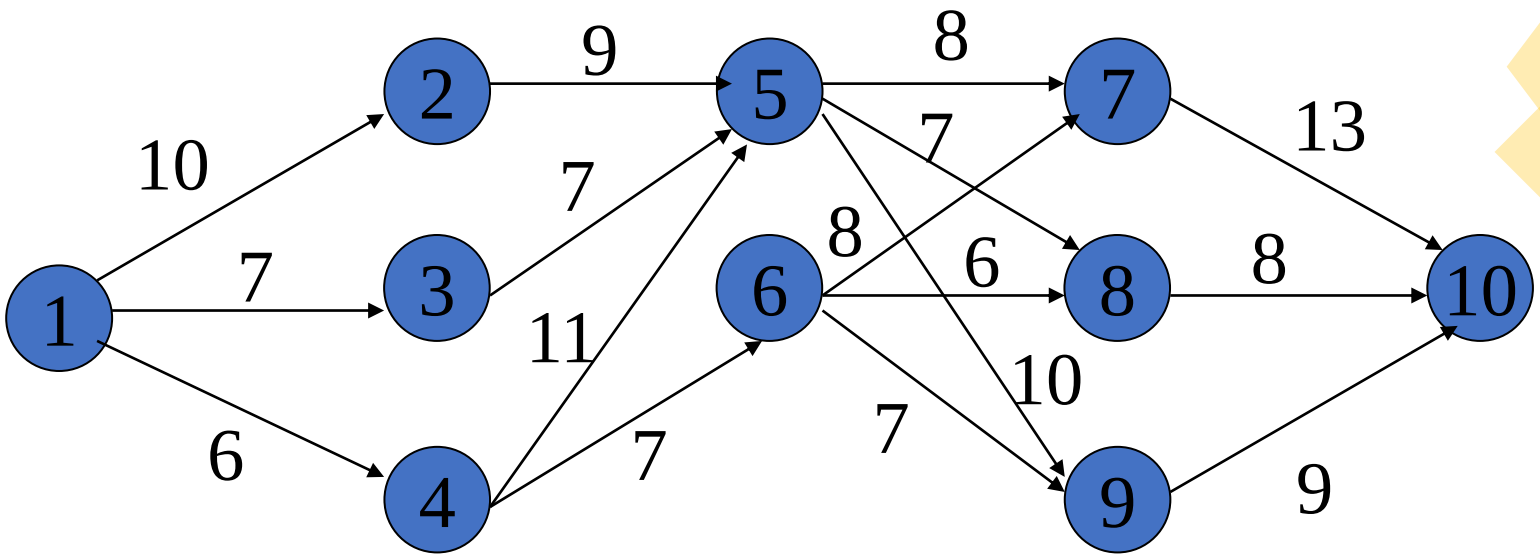
$$v^{(t)}[2] \leftarrow v^{(t-1)}[4] - 25$$
$$v^{(t)}[3] \leftarrow v^{(t-1)}[2] + 3$$
$$v^{(t)}[4] \leftarrow v^{(t-1)}[3] + 12$$



An Example

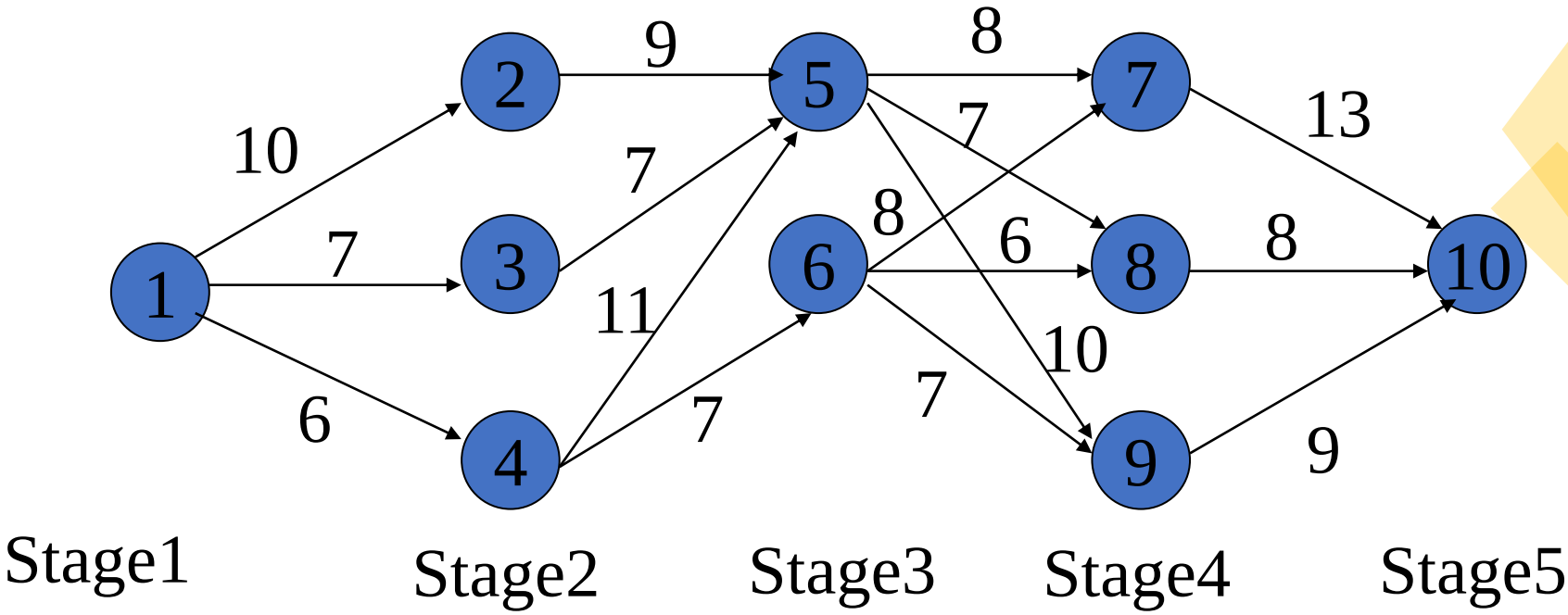


Network Example



- Numbers on arcs (c_{ij}'s) correspond to altitudes G=(N,A)
- Find a path from node 1 to node 10 such that the highest altitude on this path is the minimum among the highest altitudes of all paths

Network Example



- DP Function:
  $f_t(i)$=the highest altitude of a path with the minimum highest altitude from city i in stage t to city 10
- Optimal Value: $f_1(1)$

Network Example

- **Boundary Conditions:** $f_4(7) = 13$

  $f_4(8) = 8$

  $f_4(9) = 9$

- **Recursion using principle of optimality:**

$$f_t(\mathrm{i}) = \min_{\mathrm{j}:\,(\mathrm{i},\mathrm{j})\in A} \{\max(c_{ij}\ ,\ f_{t+1}(\mathrm{j}))\} \quad t=1,2,3$$

- **To trace:** Let d (i) be the node chosen

Network Example

$f_3(5)=8$ d(5)=8

$f_3(6)=8$ d(6)=8

$f_2(2)=9$ d(2)=5

$f_2(3)=8$ d(3)=5

$f_2(4)=8$ d(4)=6

optimal value:

$f_1(1)=8$ d(1)=3 or 4

optimal solution:

$1 \rightarrow 3 \rightarrow 5 \rightarrow 8 \rightarrow 10$

highest altitude: 8

or

$1 \rightarrow 4 \rightarrow 6 \rightarrow 8 \rightarrow 10$

highest altitude: 8

Same Example: Network Not Acyclic

- Given G=(N,A), not necessarily acyclic, specified source node s, and destination node t, and each arc (i,j) with altitude c_{ij}
- Solve the minimum highest altitude problem from s to t

- **DP Function:**

$f_k^i \equiv$ best path's highest altitude encountered in going from node i to node t among all paths that use at most k arcs

- **Optimal Value:** f_{n-1}^s where |N|=n

Same Example: Network Not Acyclic

- **Boundary Conditions:**

$$f_1^i = \begin{cases} c_{it} & \text{if} \quad (i,t) \in A \\ \infty & \text{otherwise} \end{cases} \qquad i \neq t$$

$$f_{k+1}^i = \min_{j:(i,j)\in A} \{\max(c_{ij}, f_k^j)\} \qquad k = 1, \ldots, n-2$$

- **To trace:** Let d (i) be the node chosen