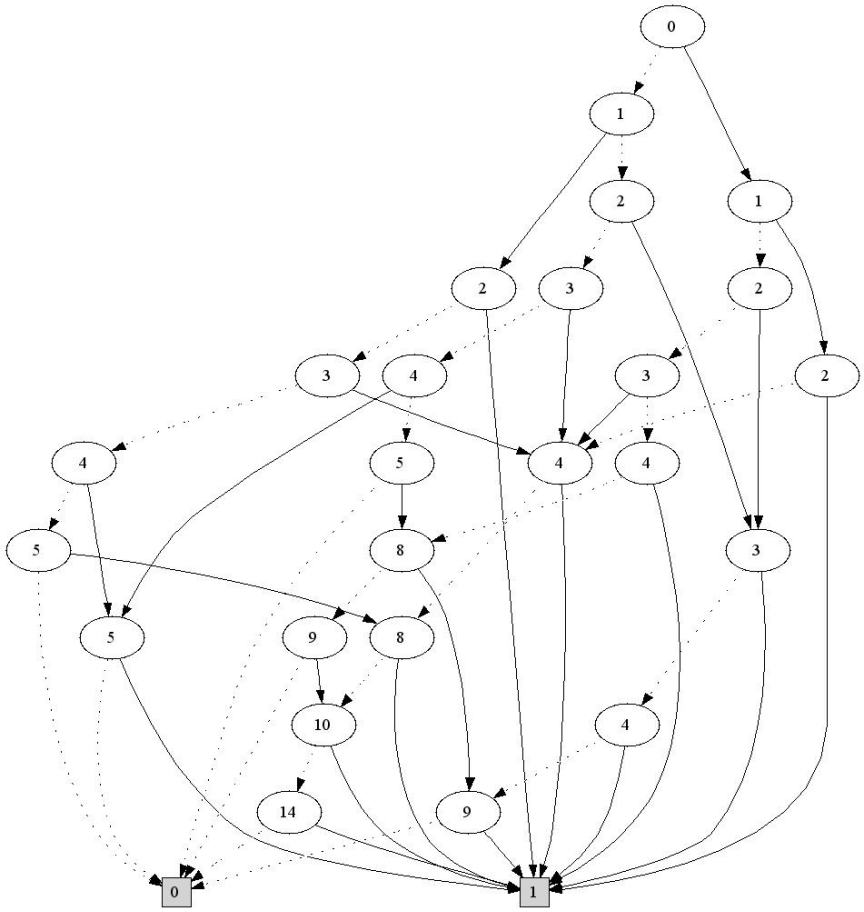
# Decision Diagrams and Dynamic Programming

J. N. Hooker
Carnegie Mellon University

CPAIOR 2013

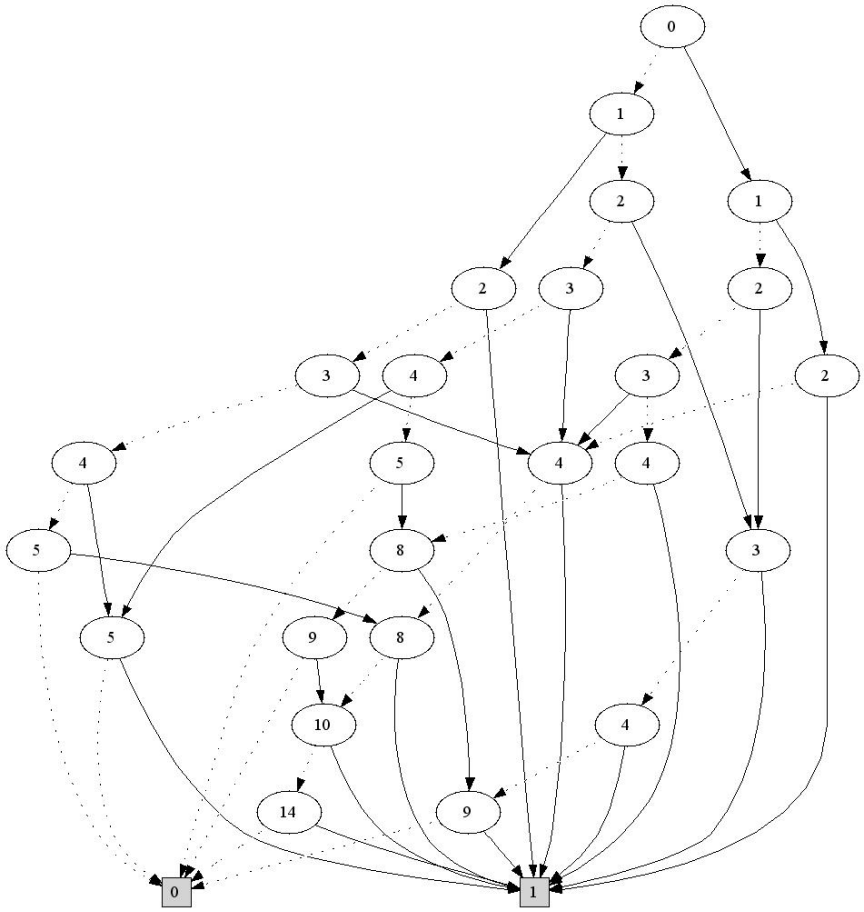
# Decision Diagrams & Dynamic Programming

- Binary/multivalued **decision diagrams** are related to **dynamic programming**.
  - But there are important differences.



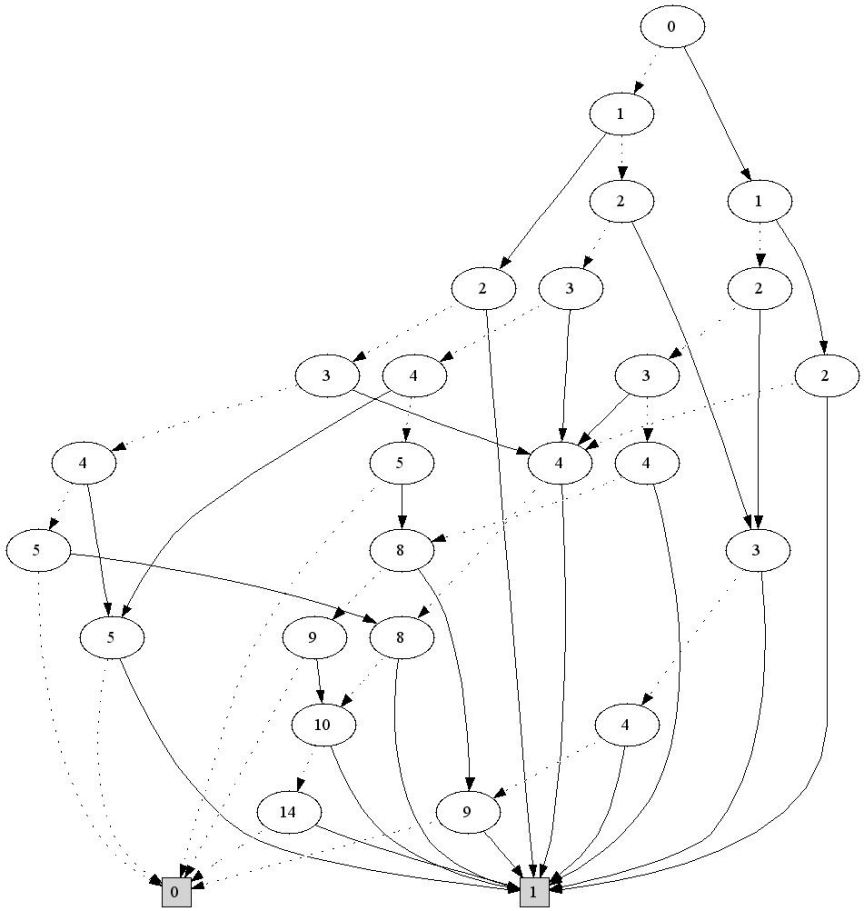
# Decision Diagrams & Dynamic Programming

- Binary/multivalued **decision diagrams** are related to **dynamic programming**.
  - But there are important differences.
  - Dynamic programming has **state variables** and **state-dependent costs**.



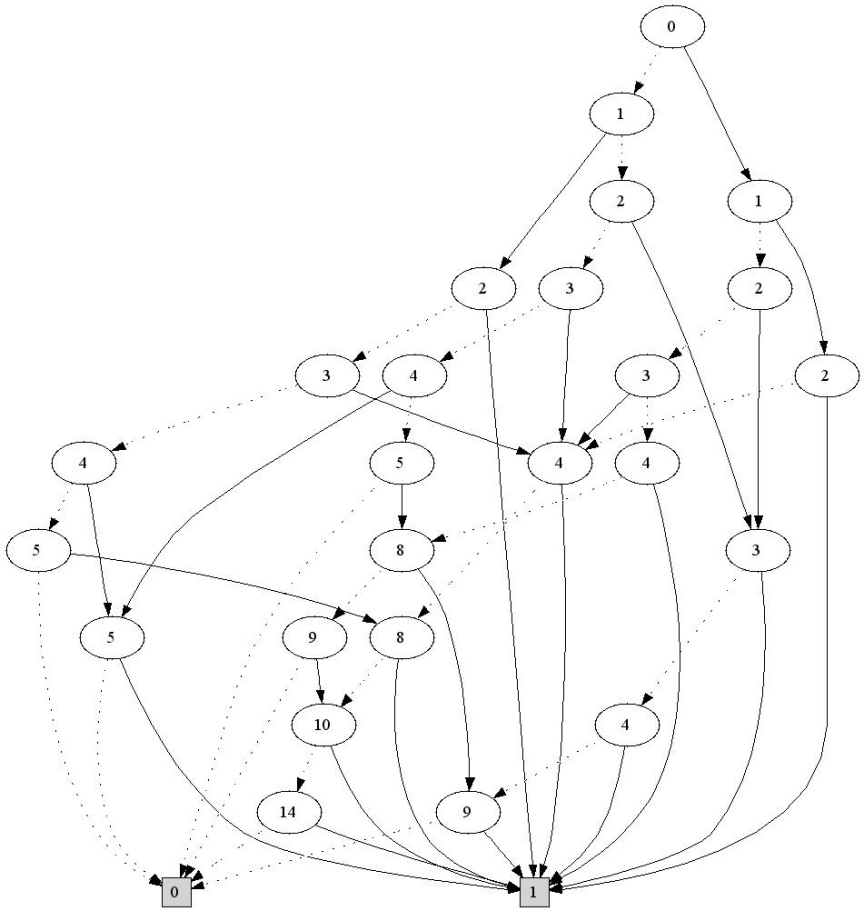
# Decision Diagrams & Dynamic Programming

- We **extend** the theory of decision diagrams to accommodate state-dependent-costs.
  - We prove **uniqueness** theorem for weighted DDs using canonical costs.



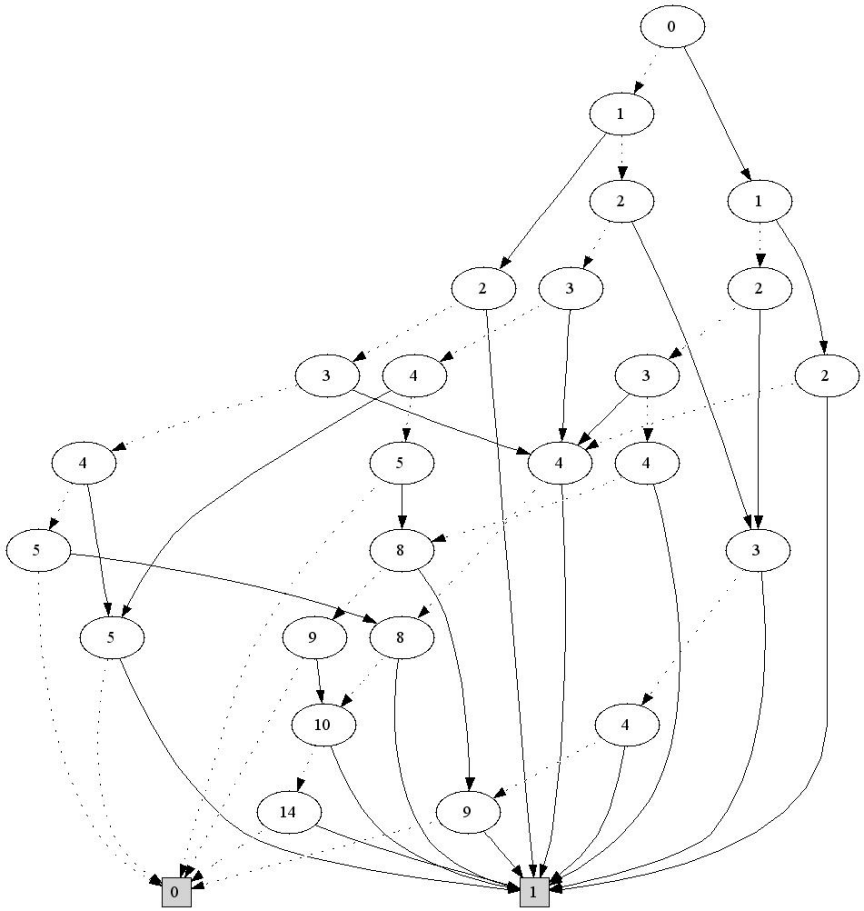
# Decision Diagrams & Dynamic Programming

- We **extend** the theory of decision diagrams to accommodate state-dependent-costs.
  - We prove **uniqueness** theorem for weighted DDs using canonical costs.
- We can now view DP state transition graph as a decision diagram.
  - And perhaps **reduce** the decision diagram to simplify the DP model.



# Outline

- Dynamic programming example
- Weighted decision diagrams and canonical costs
- Application to the example
- Future work



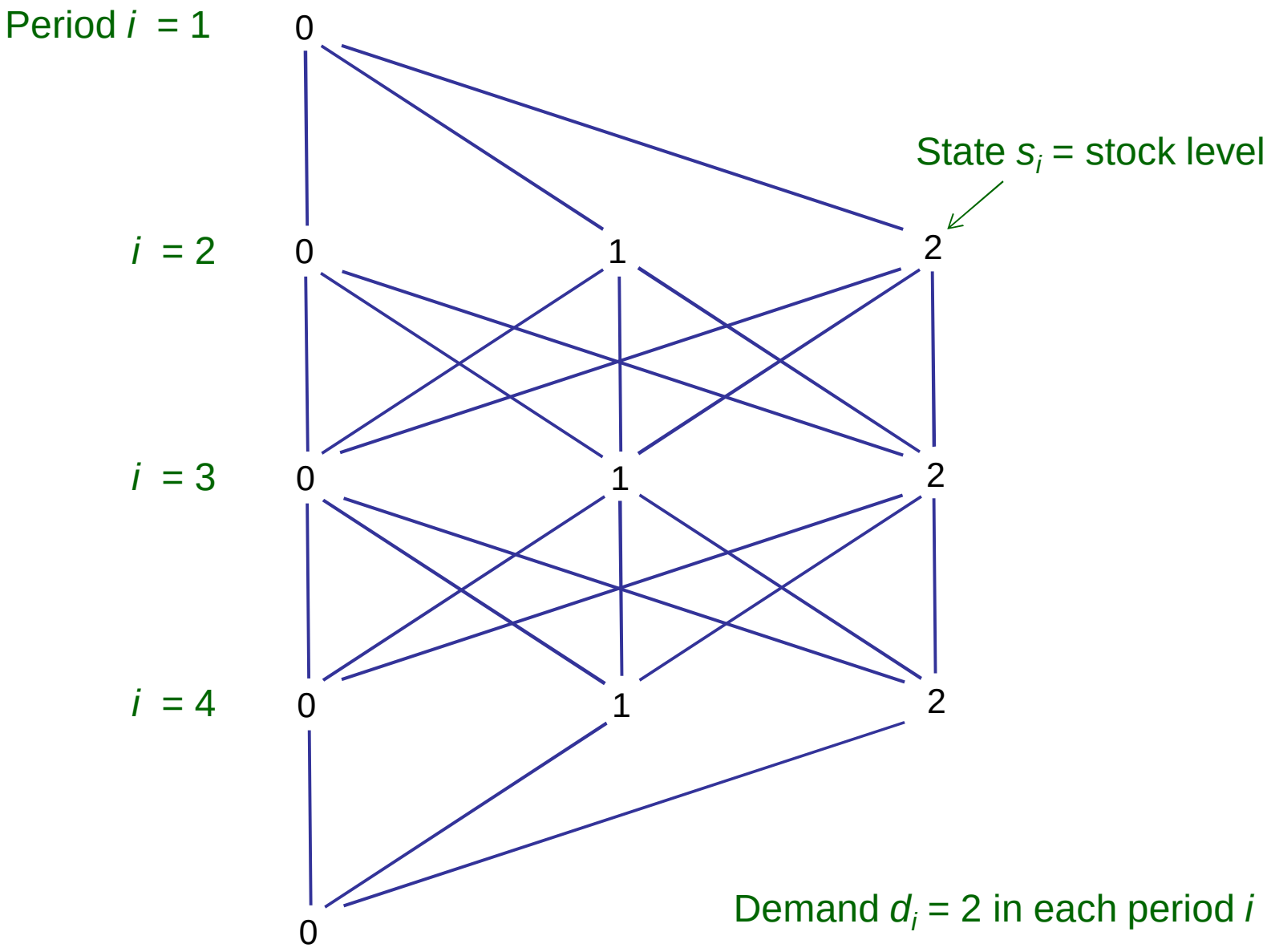
# Dynamic Programming

- **Dynamic programming** (including the name) was introduced by Richard Bellman in 1950s.
  - Different concept than decision diagram, caching, etc.
  - But DP state transition graph can be viewed as a weighted decision diagram.
- Illustration: a very basic inventory management problem.
  - In the literature at least 50 years.

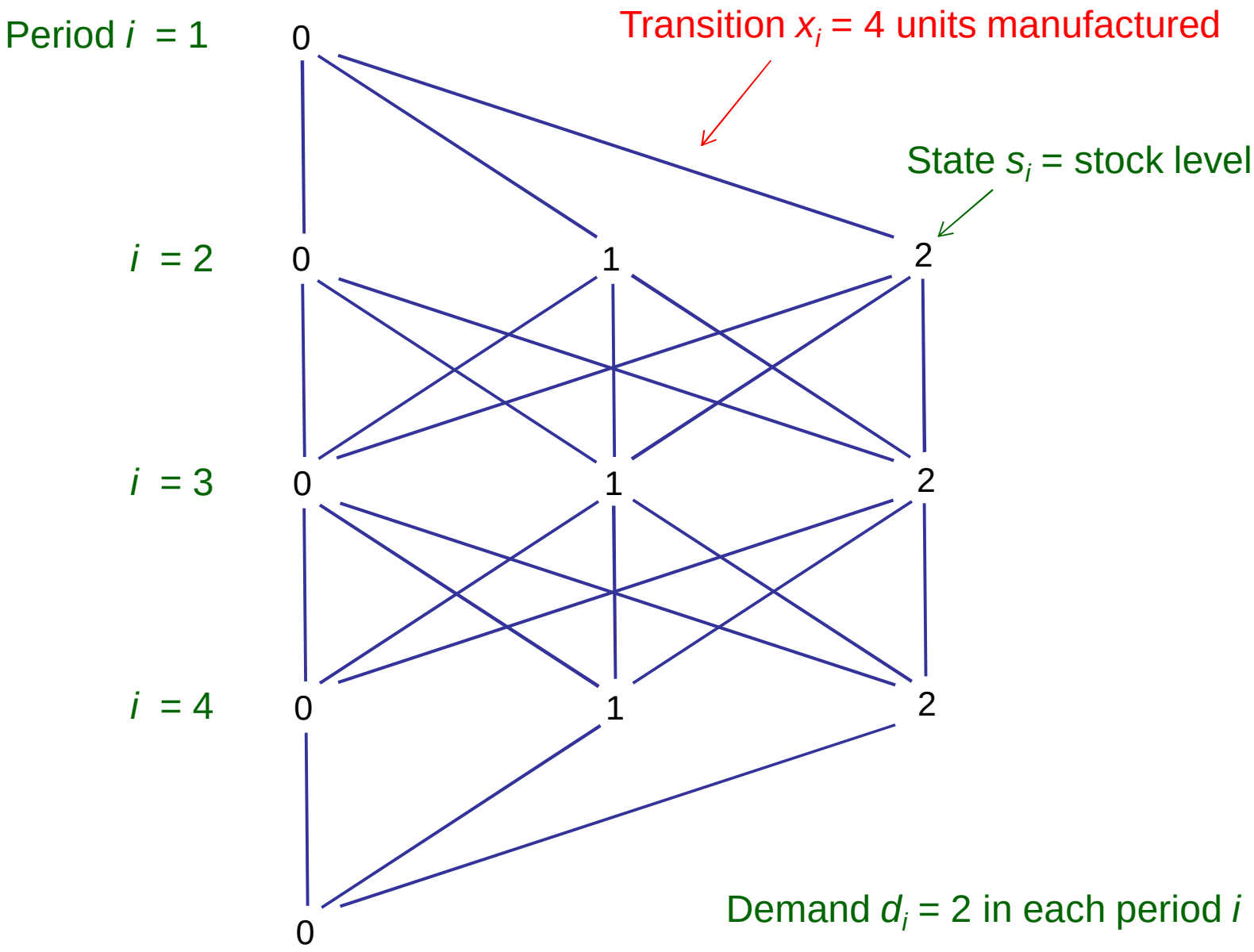
# Inventory Management Example

- In each period $i$, we have:
  - Demand $d_i$.
  - Unit production cost $c_i$.
  - Warehouse space $m$.
  - Unit holding cost $h_i$.
- In each period, we decide:
  - Production level $x_i$.
  - Stock level $s_i$.
- Objective:
  - Meet demand each period while minimizing production and holding costs.

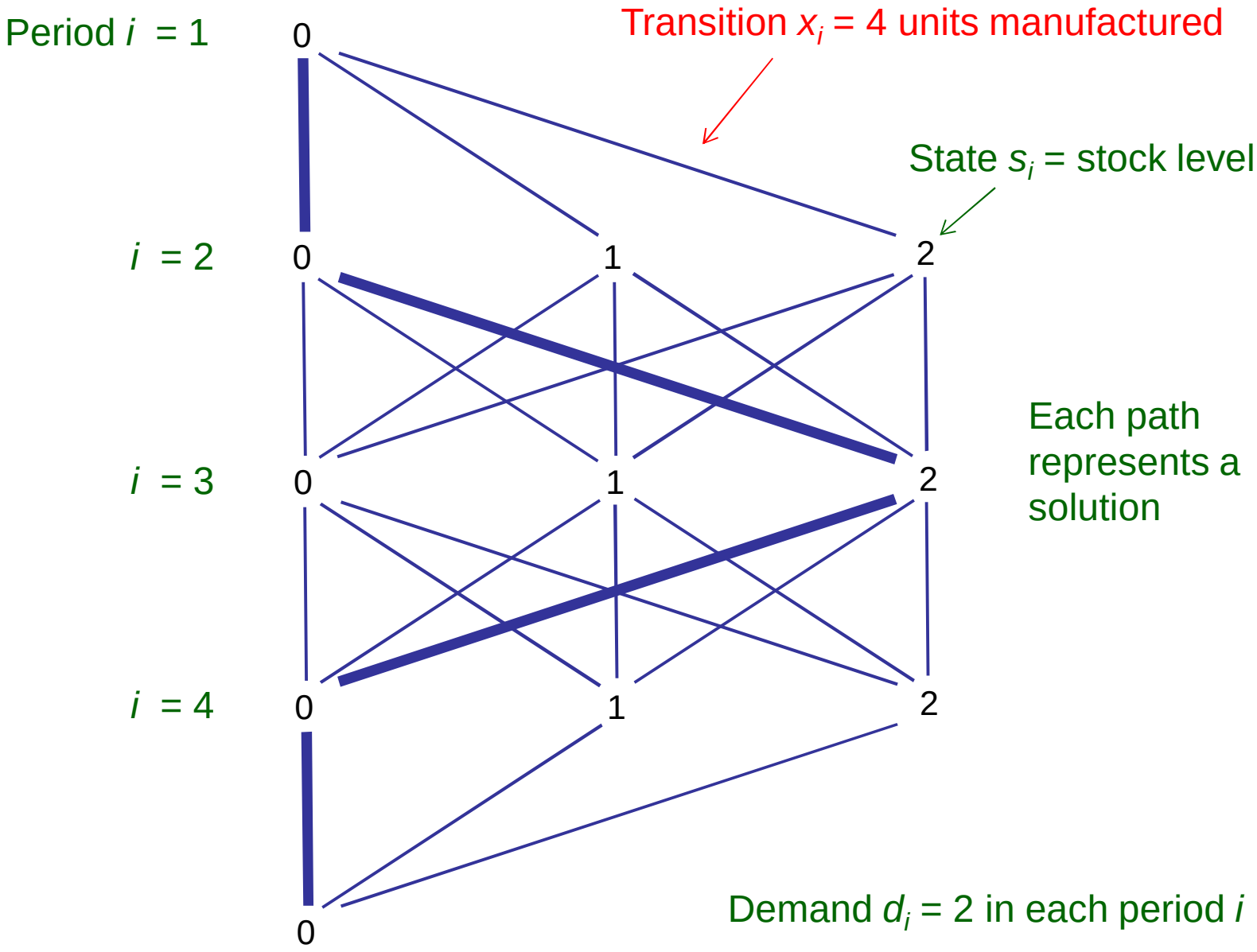
# State transition graph



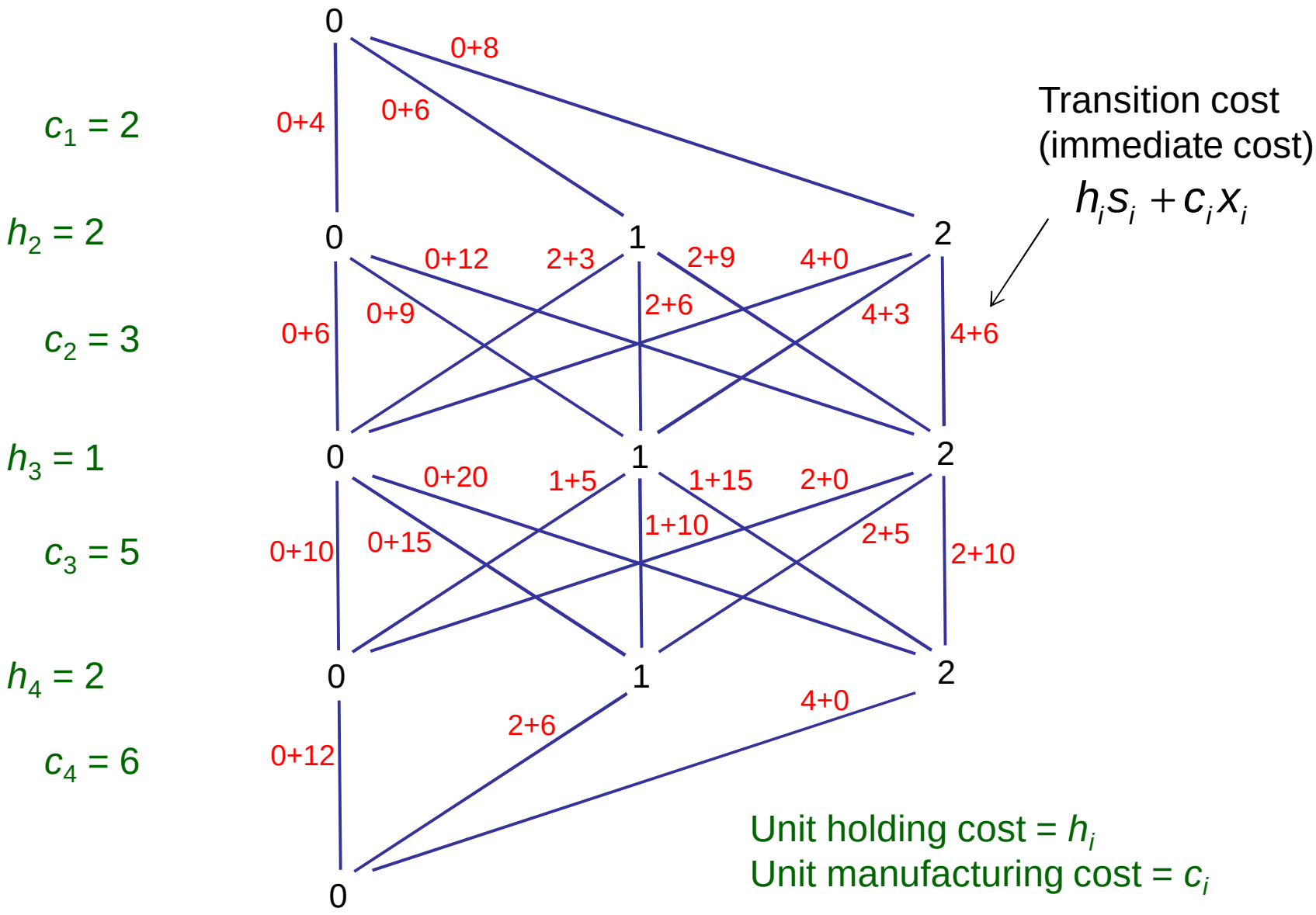
# State transition graph



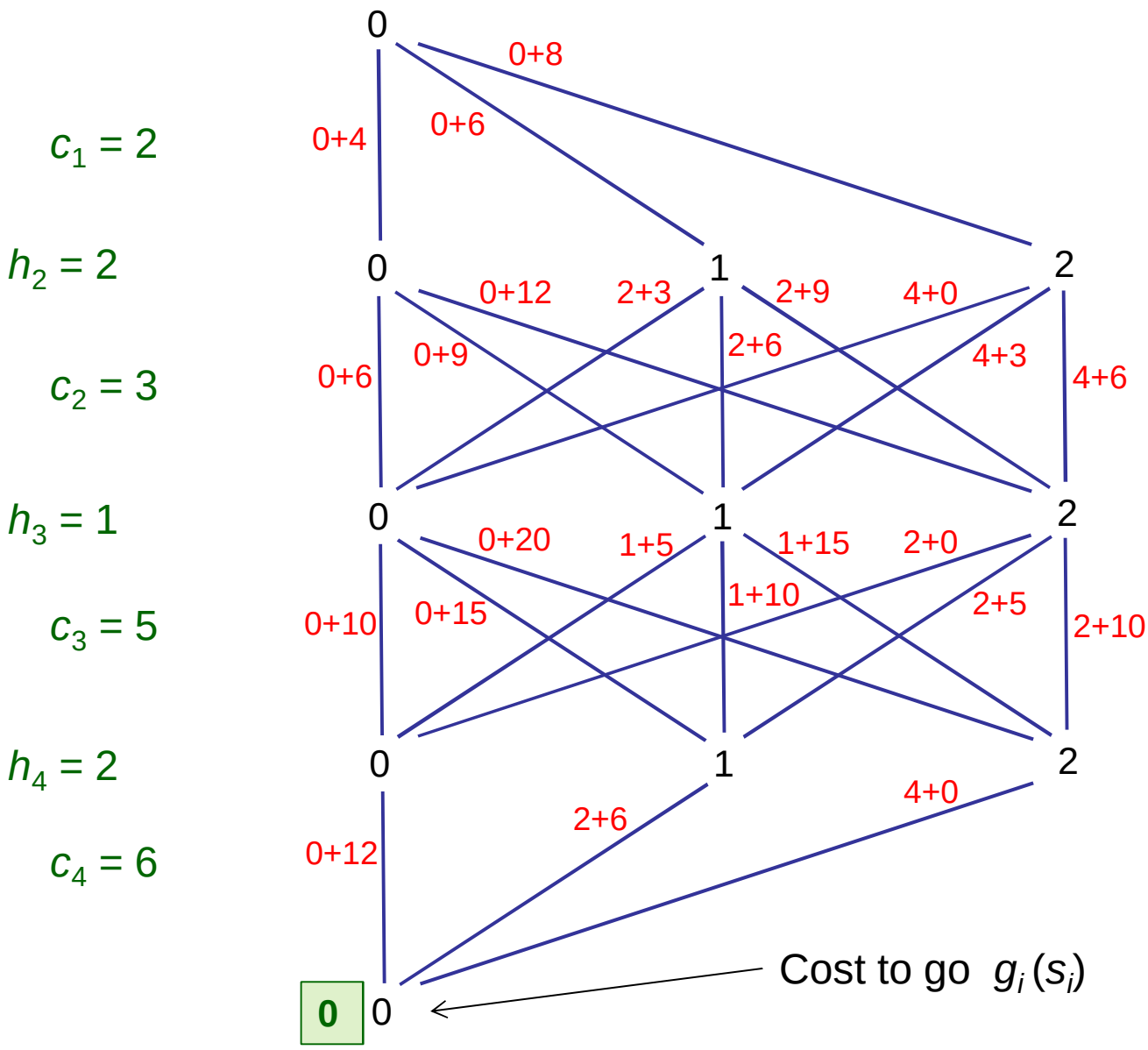
# State transition graph



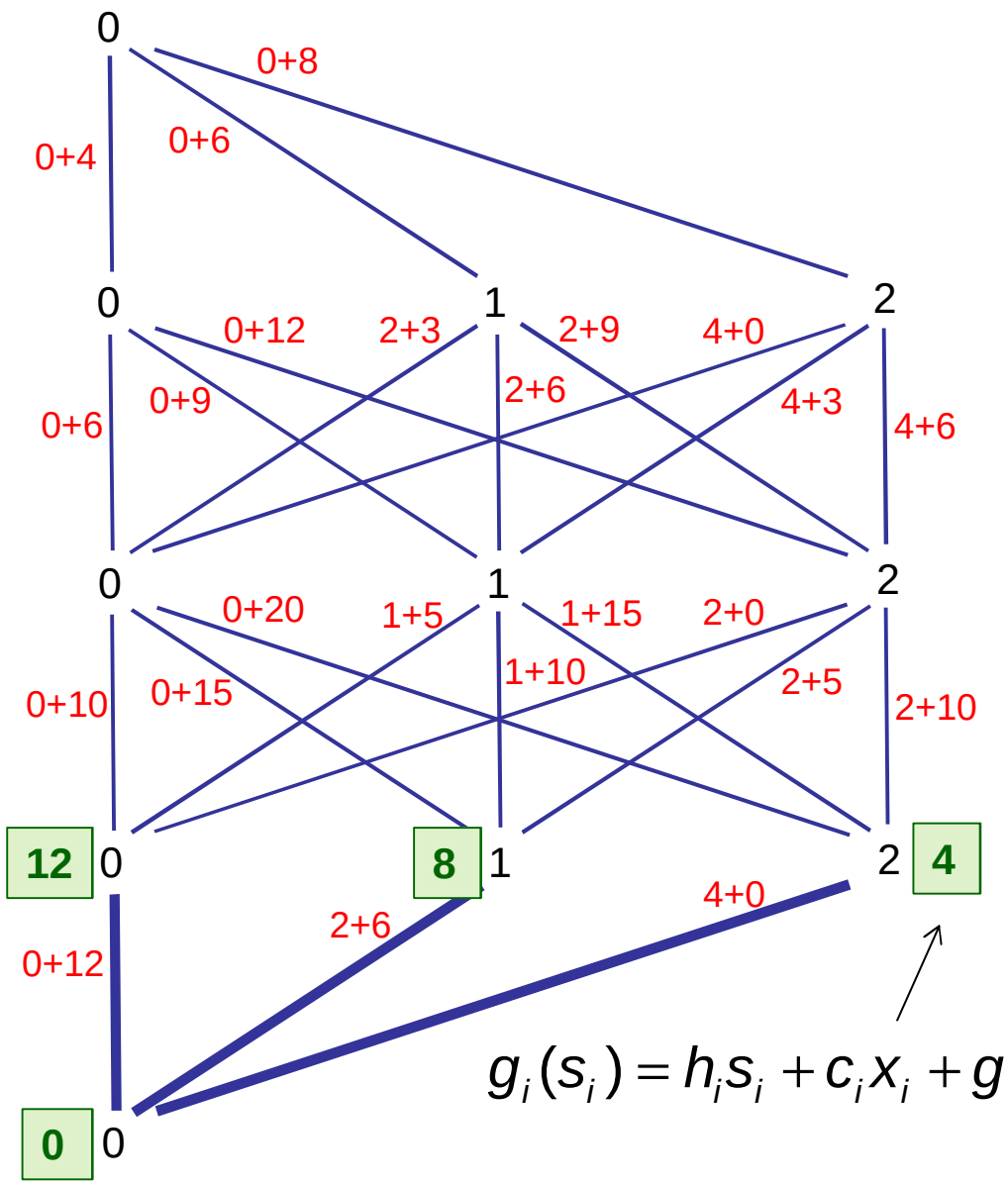
# Transition costs



# Backward recursion

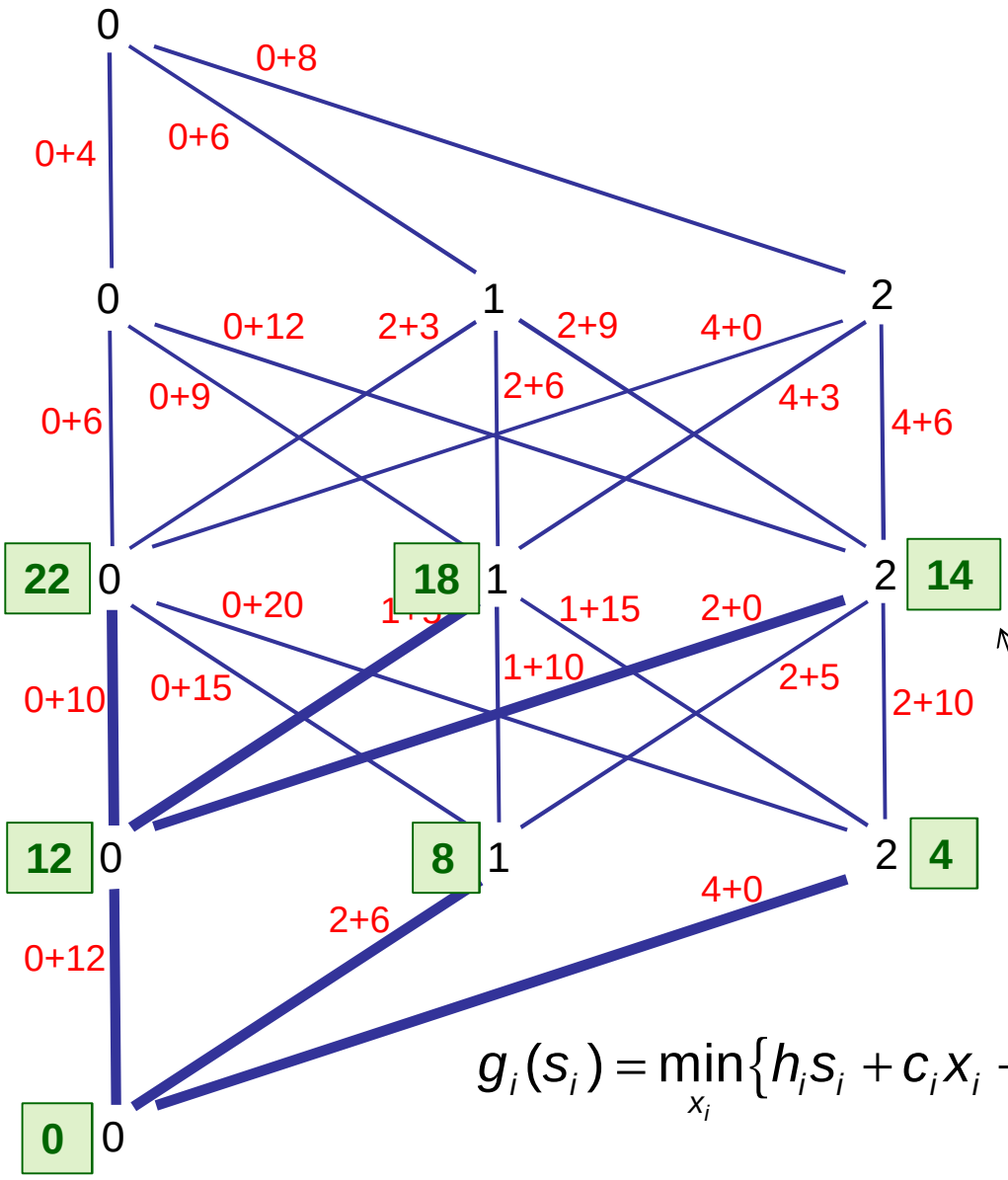


# Backward recursion



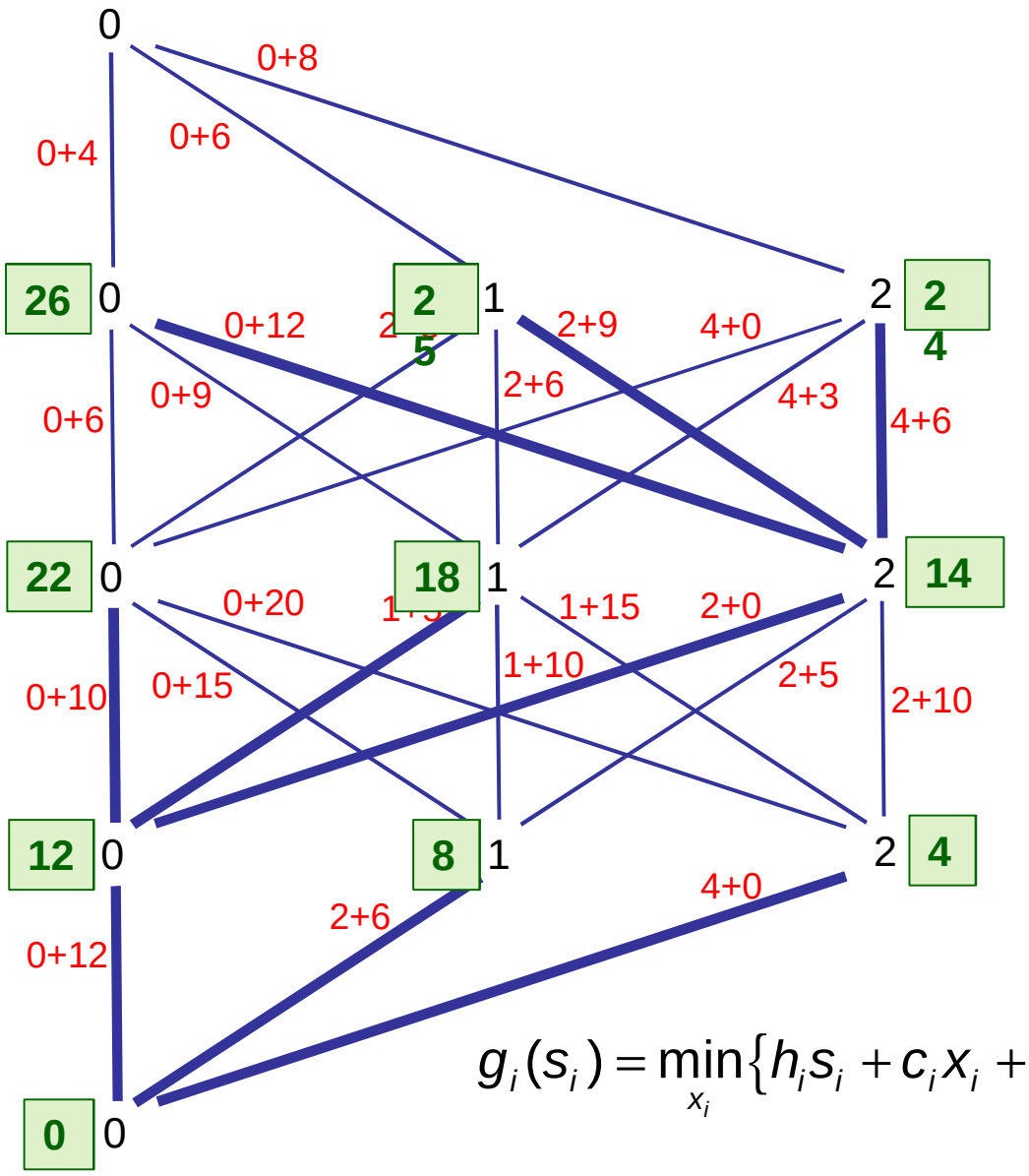

$$g_i(s_i) = h_i s_i + c_i x_i + g_{i+1}(s_i + x_i - d_i)$$

# Backward recursion



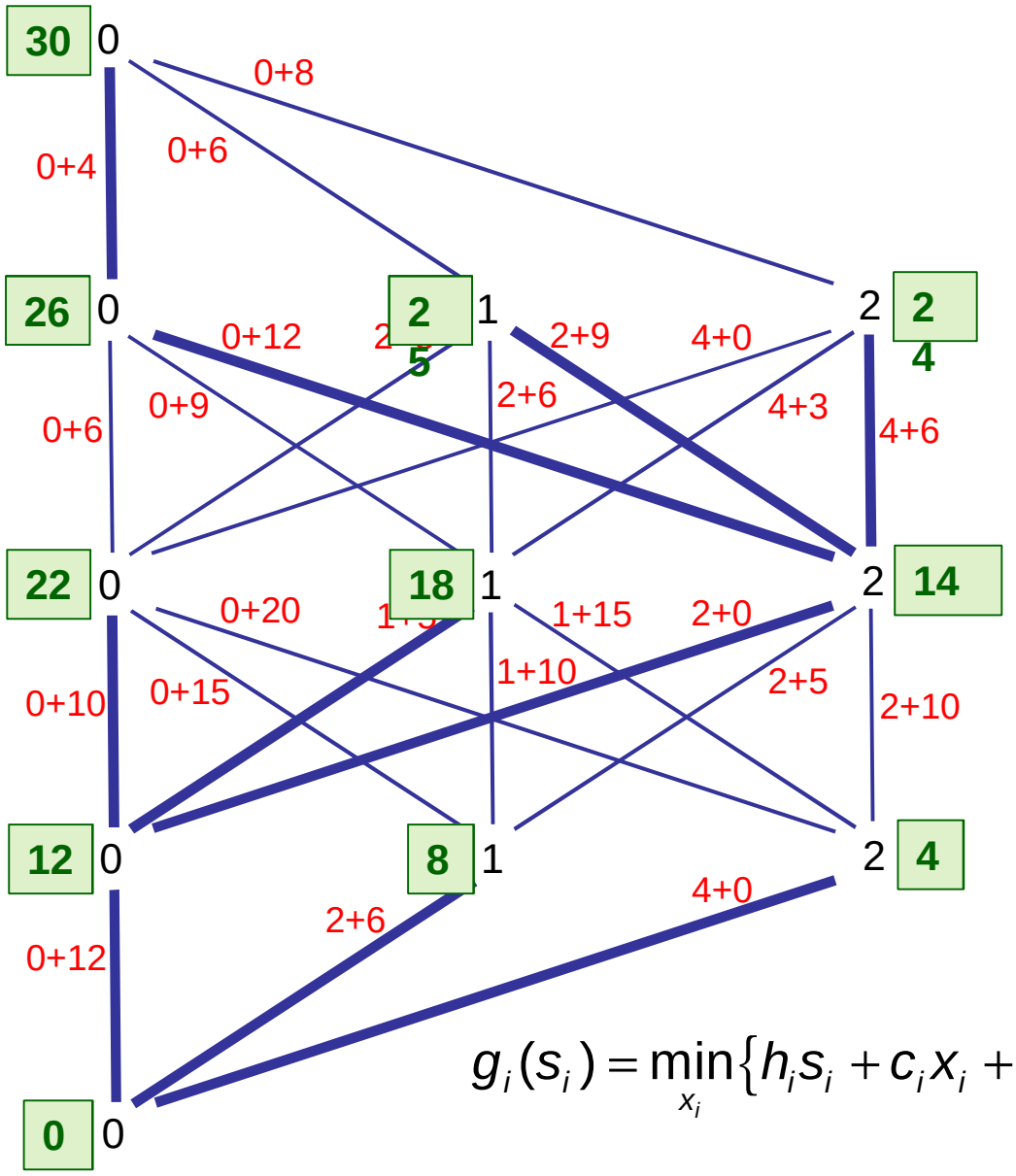

$$g_i(s_i) = \min_{x_i}\{h_i s_i + c_i x_i + g_{i+1}(s_i + x_i - d_i)\}$$

# Backward recursion



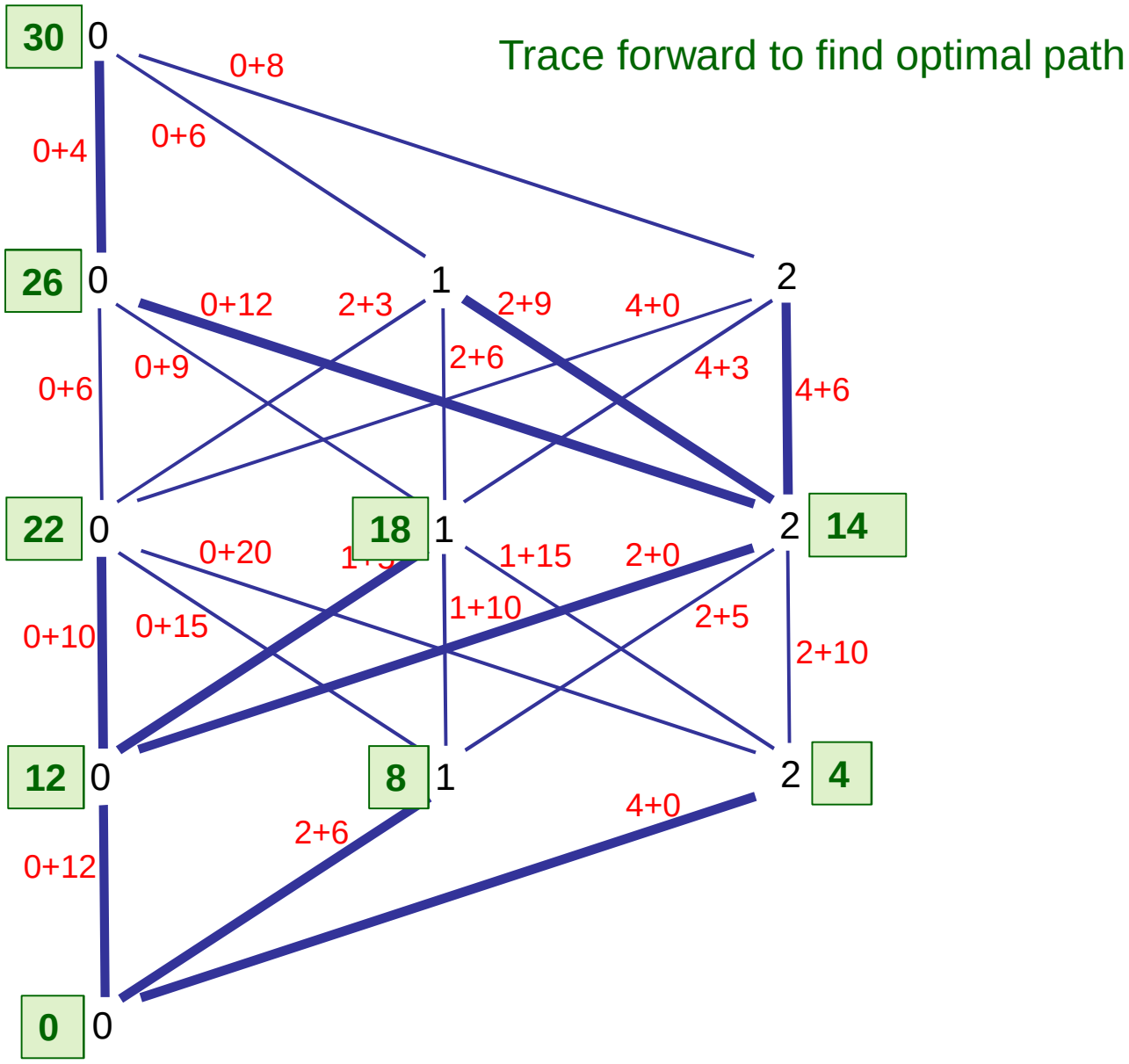

$$g_i(s_i) = \min_{x_i}\{h_i s_i + c_i x_i + g_{i+1}(s_i + x_i - d_i)\}$$

# Backward recursion




$$g_i(s_i) = \min_{x_i}\{h_i s_i + c_i x_i + g_{i+1}(s_i + x_i - d_i)\}$$

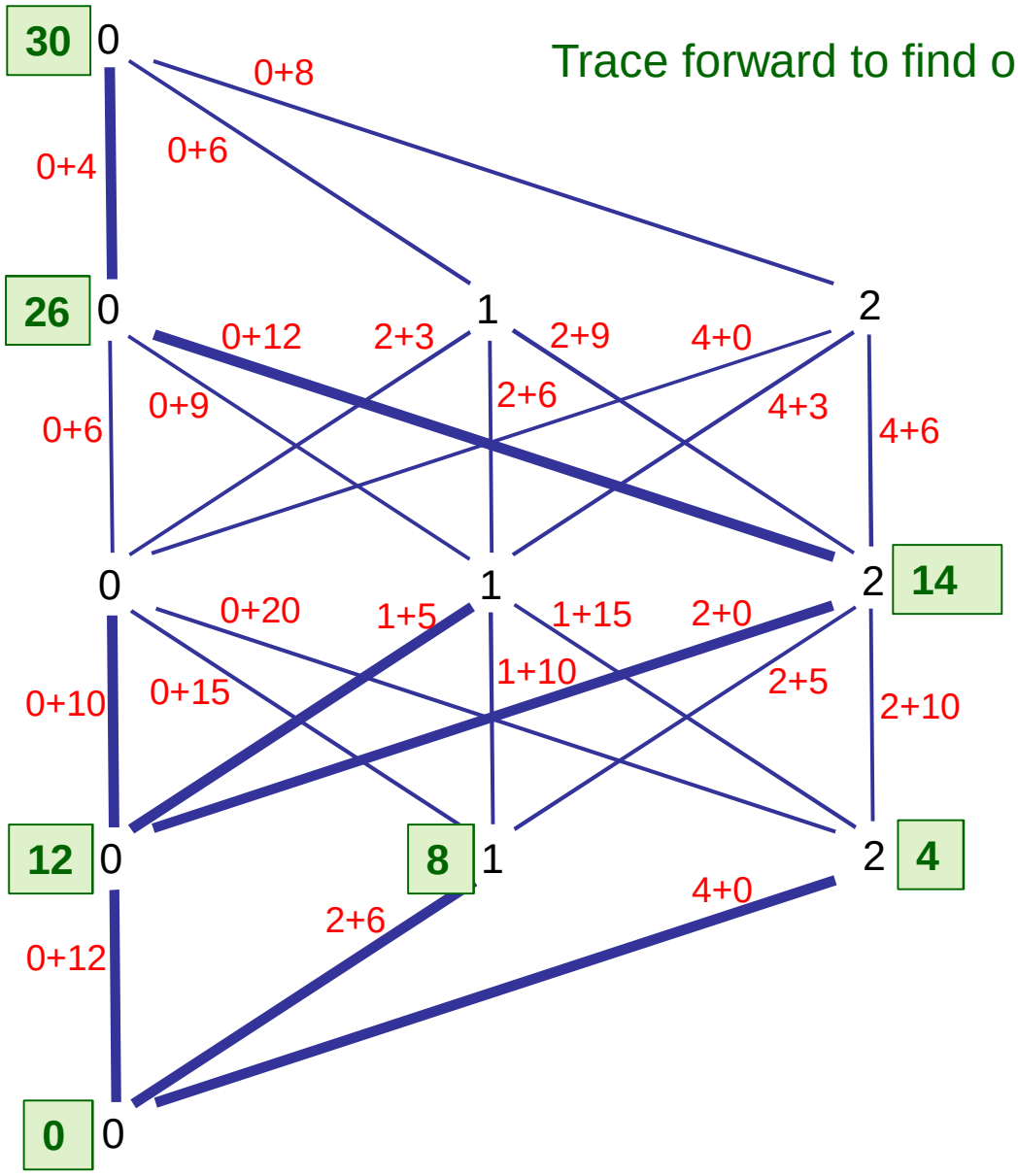
# Optimal solution



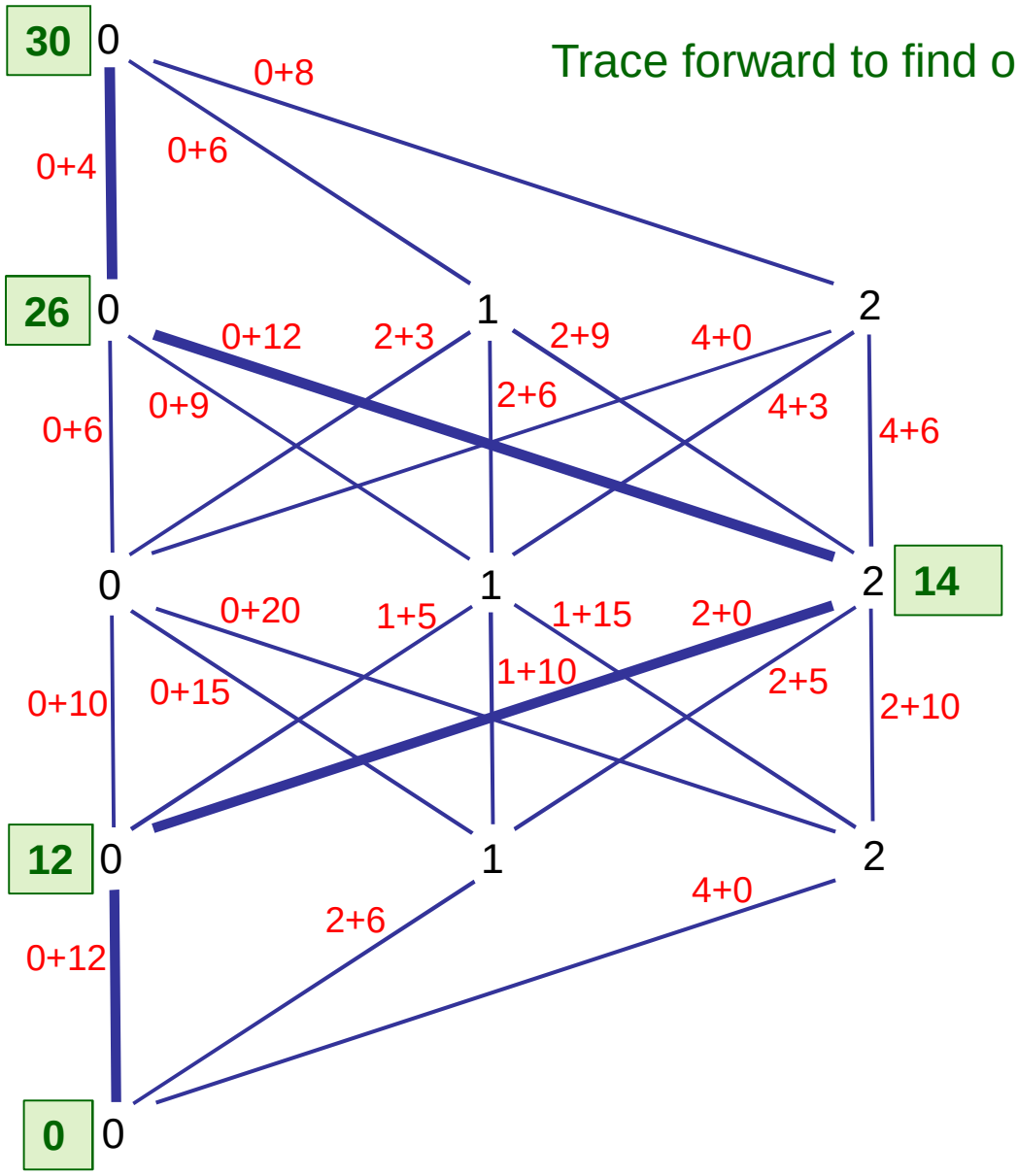

Trace forward to find optimal path

# Optimal solution

Trace forward to find optimal path



# Optimal solution




Trace forward to find optimal path

# Dynamic Programming Recursion

- In general, the state transition is

$$s_{i+1} = \varphi_i(s_i, x_i), \quad i = 1, \ldots, n$$

- Cost is a function of state and control pairs

$$f(x) = \sum_{i=1}^{n} c_i(s_i, x_i)$$

- The recursion is

$$g_i(s_i) = \min_{x_i} \left\{ c_i(s_i, x_i) + g_{i+1}\left(\varphi_i(s_i, x_i)\right) \right\}$$

  - with boundary condition $g_{n+1}(s_{n+1}) = 0, \quad \text{all } s_{n+1}$

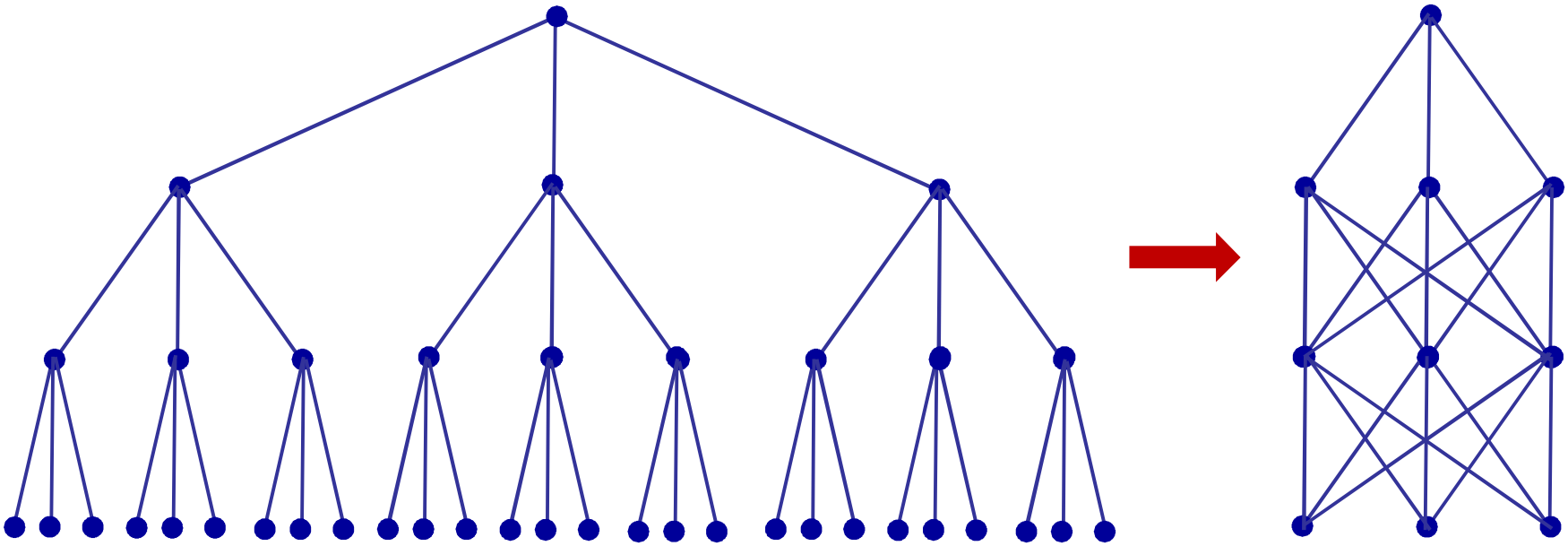
  - and optimal value $g_1(s_1)$ for starting state $s_1$

# Dynamic Programming Characteristics

- There are **state variables** in addition to decision variables.
- **Costs** are function of **state variables** as well as decision variables.
- State transitions are **Markovian**.
  - Current state determines possible transitions and costs.
- Problem is solved **recursively**.
  - Often by moving backward through stages.
- The **art** of dynamic programming:
  - Find a small state description that is Markovian.

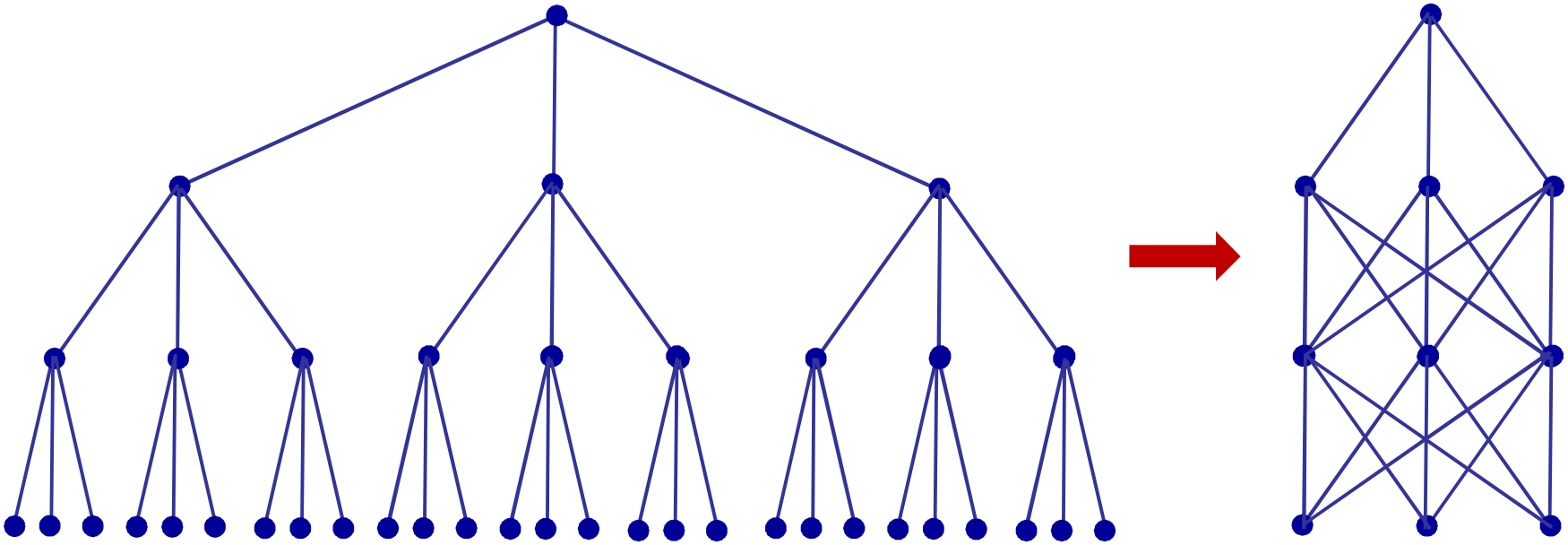
# DP vs Caching

- Dynamic programming ≠ caching
  - Yes, DP identifies equivalent subproblems.



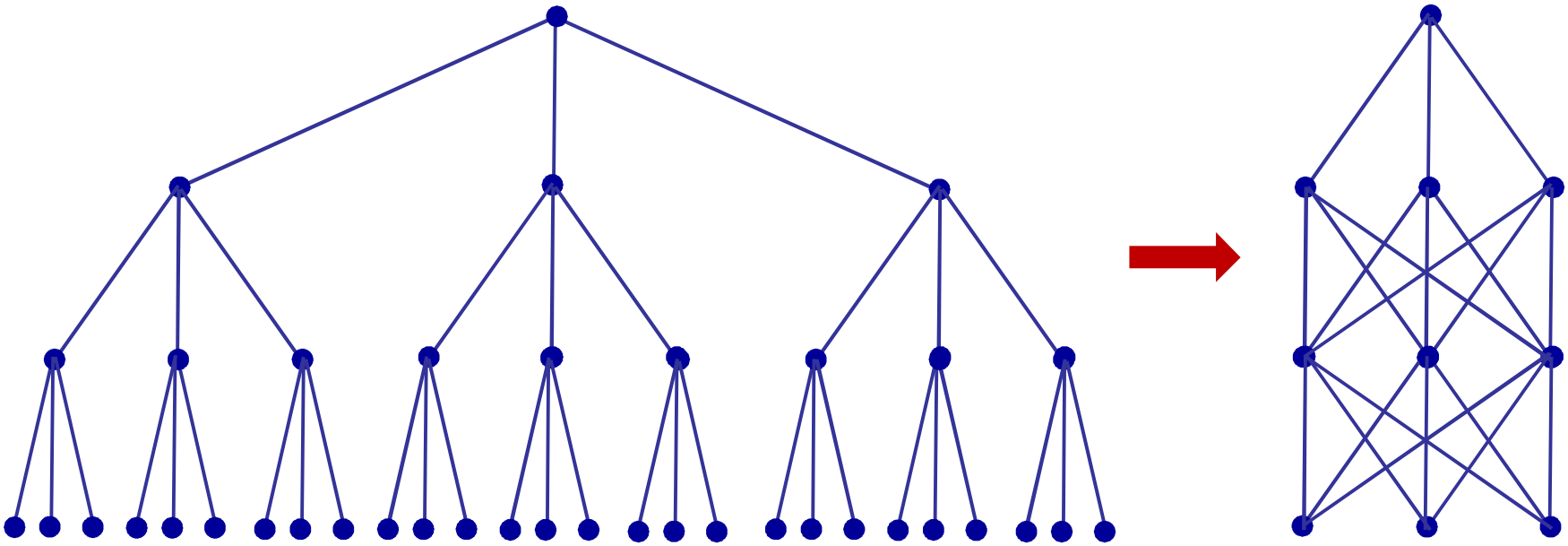
# DP vs Caching

- Dynamic programming ≠ caching
  - Yes, DP identifies equivalent subproblems.
  - But **not** by identifying equivalent states.
  - All states are treated separately (except in approximate DP).
- The intelligence is in the state description.

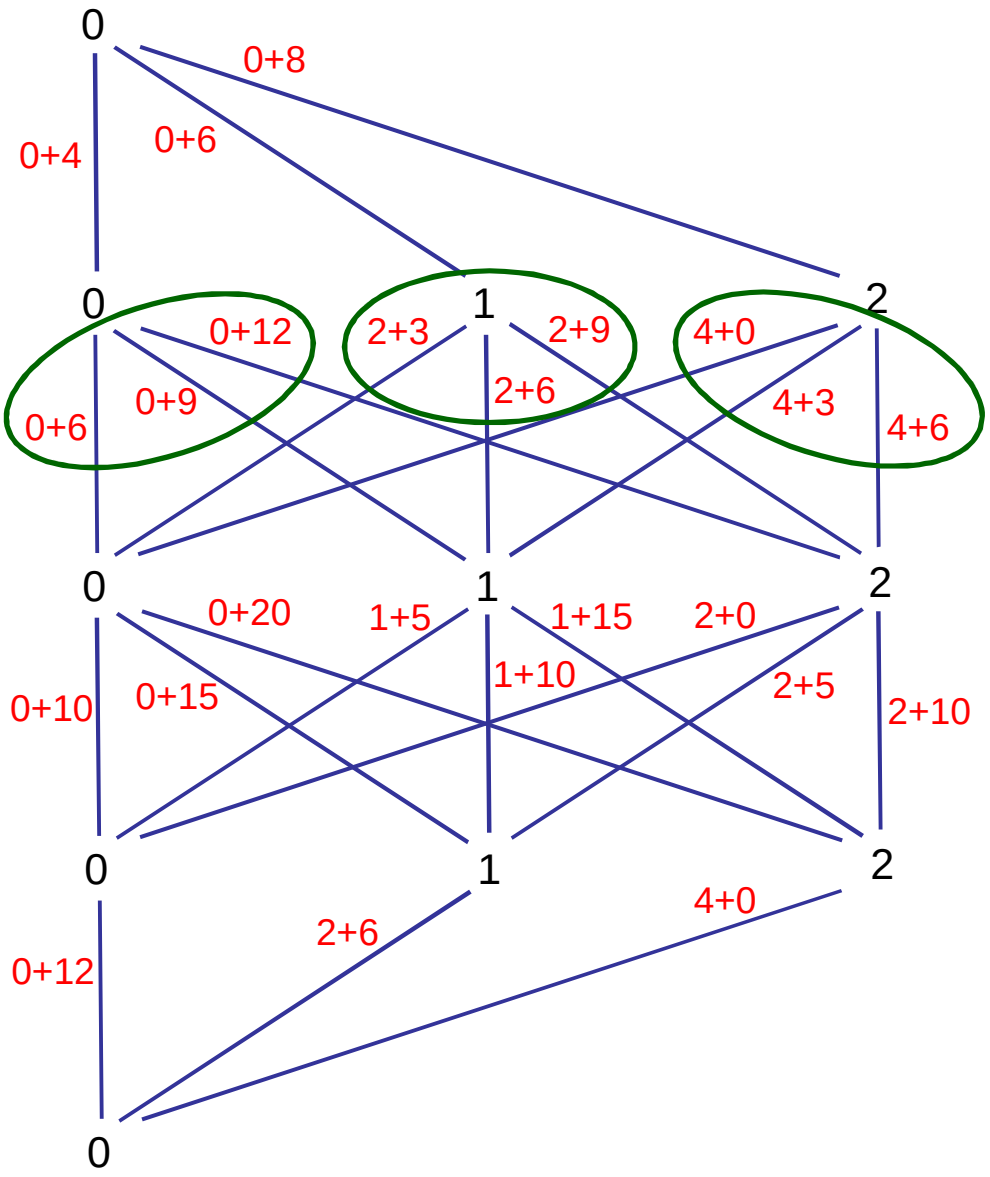


# DP vs Caching

- However, caching can be applied **on top of** DP.
  - We will use the concept of **reduced decision diagram** (reduced MDD) to identify equivalent states.
- Problem: how to deal with **state-dependent costs**.



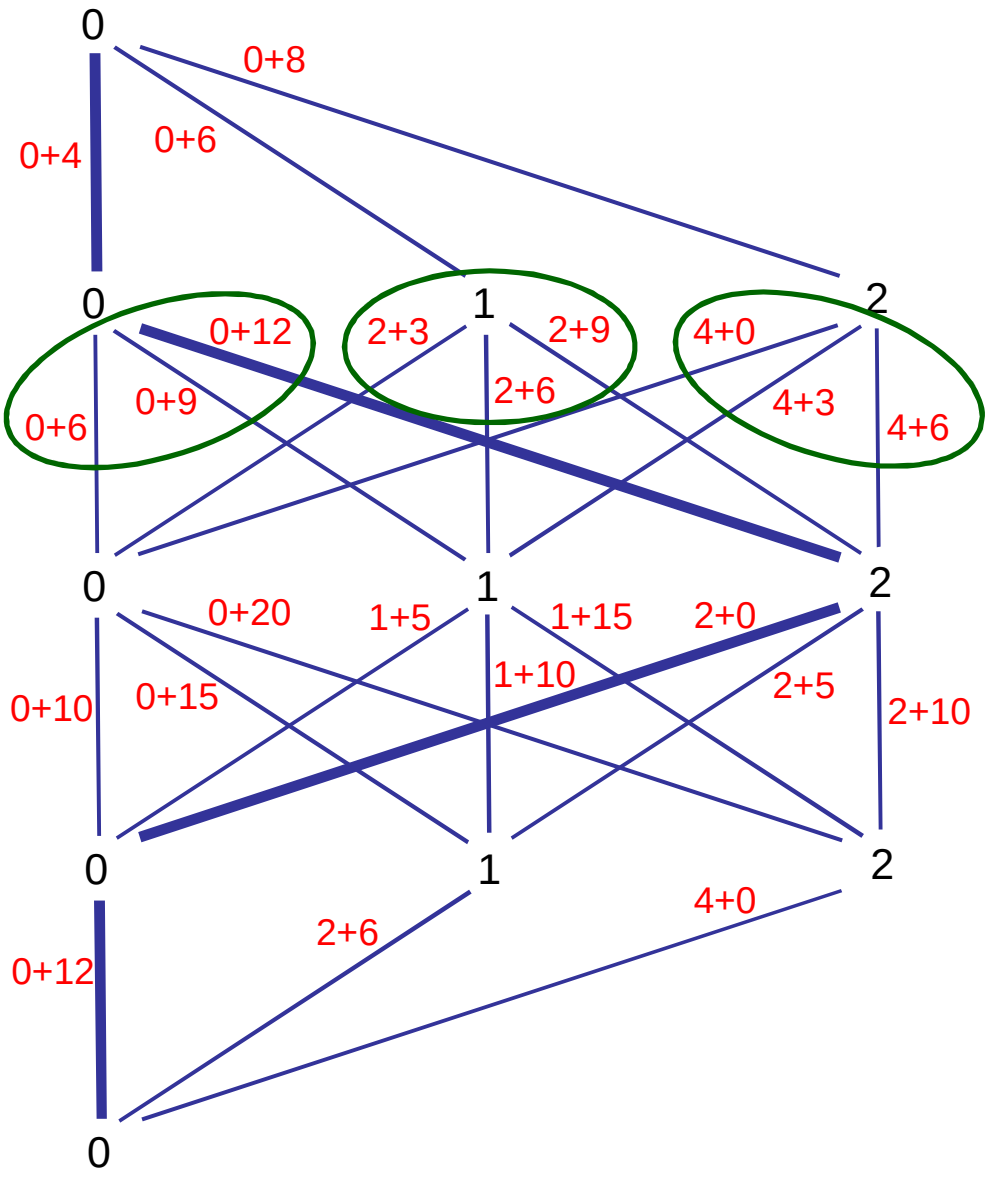
# Reducing the Transition Graph




Arcs leaving each node are very similar.

- Transition to the same states.
- Have the same costs, up to an offset.

# Reducing the Transition Graph

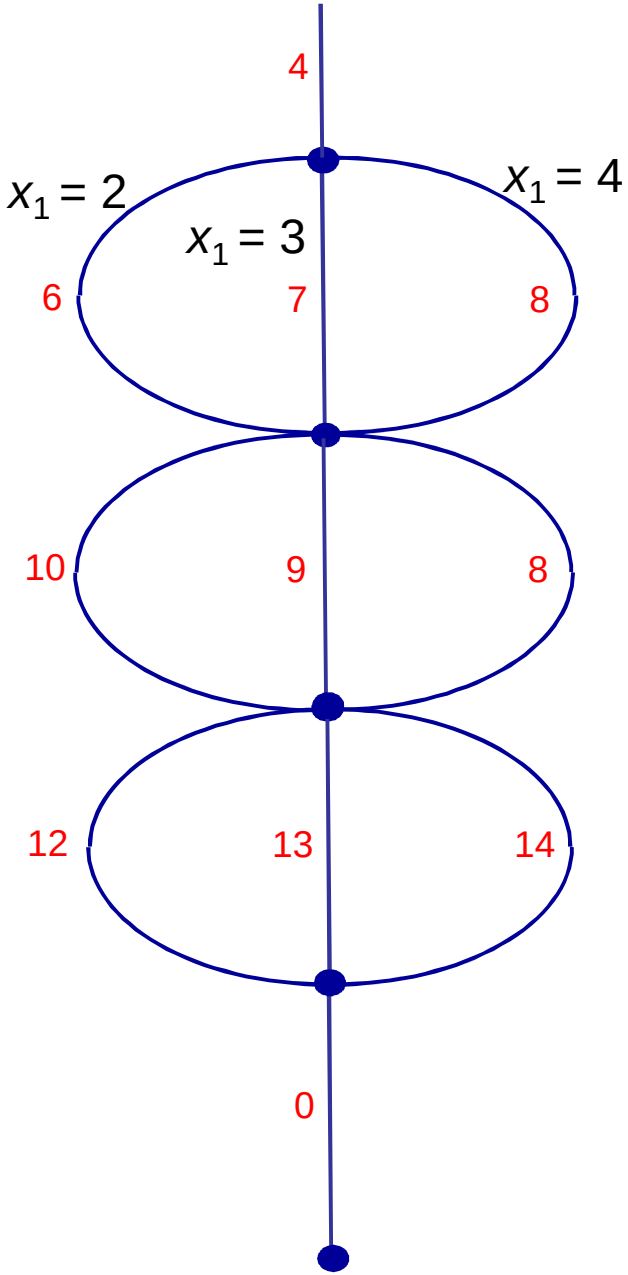



Arcs leaving each node are very similar.

- Transition to the same states.
- Have the same costs, up to an offset.

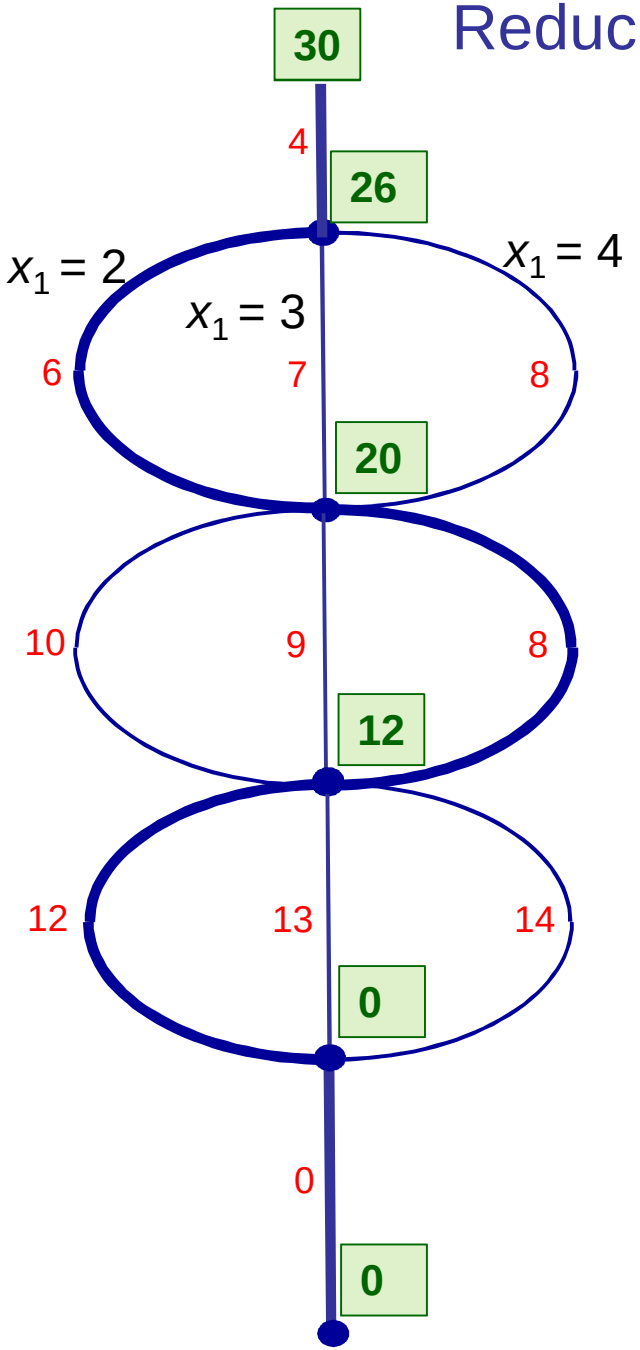
Incidentally, there is also a **bang-bang** solution.

# Reducing the Transition Graph




By rearranging the costs, we can collapse the states in each period.

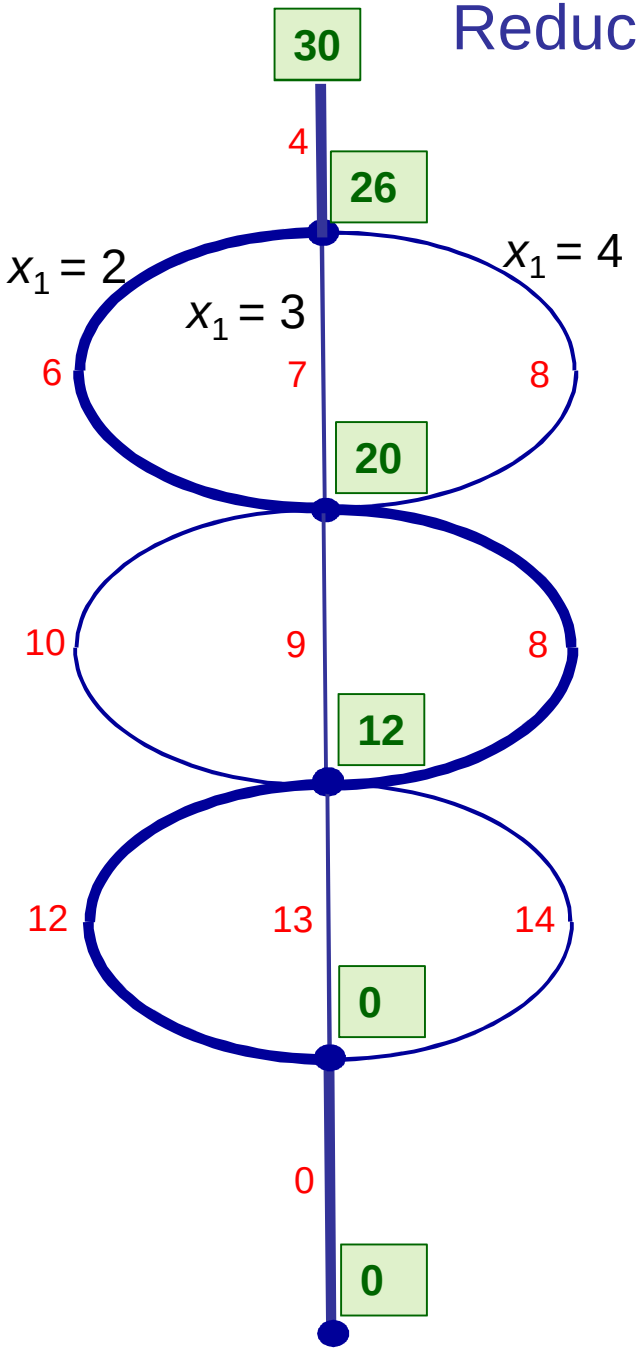
# Reducing the Transition Graph




By rearranging the costs, we can collapse the states in each period.

Now it is easier to compute the optimal solution

# Reducing the Transition Graph




By rearranging the costs, we can collapse the states in each period.

Now it is easier to compute the optimal solution

This looks like reduction of a **decision diagram** (MDD).

We will develop this idea in general.

# Decision Diagrams

**Set covering example**

Select a minimum-weight family of sets that contain all 4 elements A, B, C, D

| | Set $i$ | | | |
|---|---|---|---|---|
| | 1 | 2 | 3 | 4 |
| A | • | • | | |
| B | • | | • | • |
| C | | • | • | |
| D | | • | | • |
| Weight | 3 | 5 | 4 | 6 |

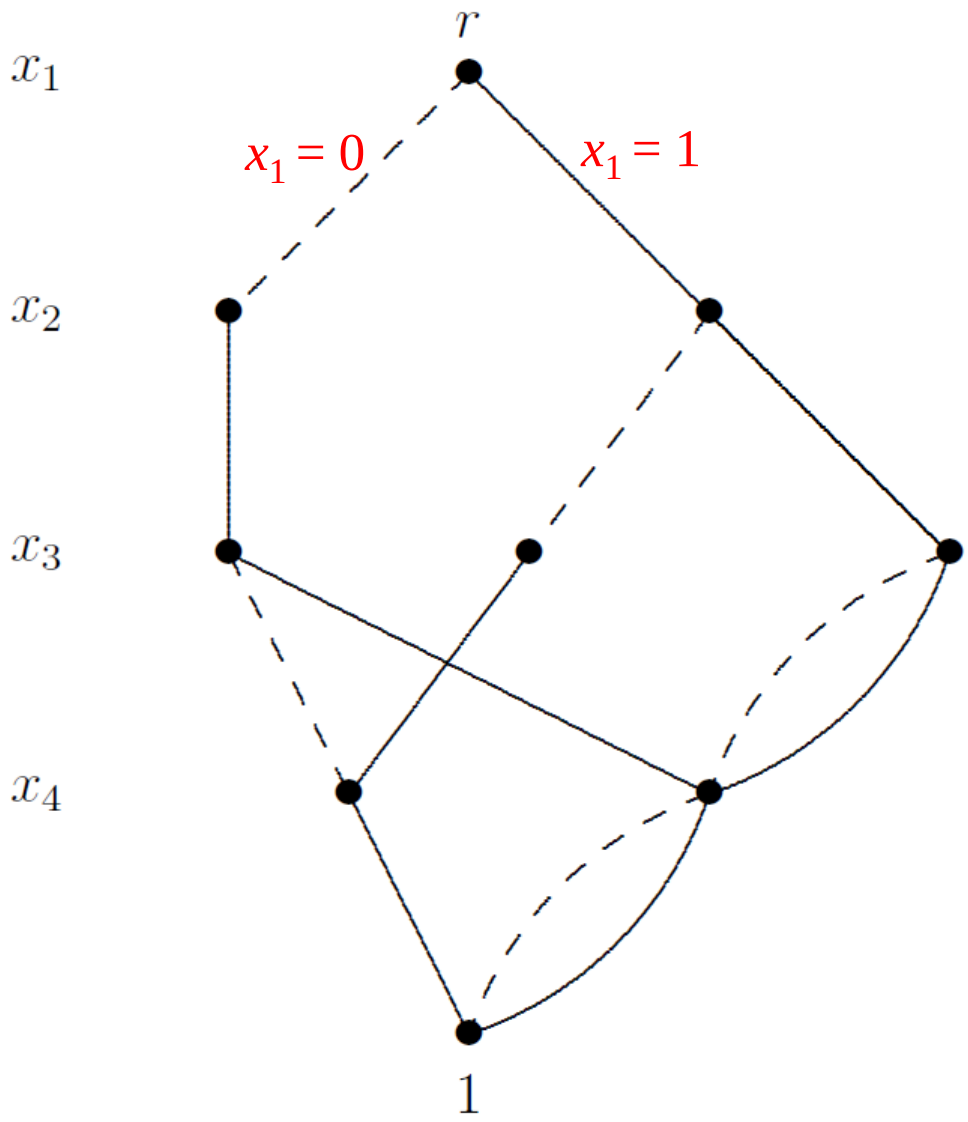
$x_i$ = 1 when we select set $i$

# Decision Diagrams

**Decision diagram**

Each path corresponds to a feasible solution.

| | Set $i$ | | | |
|---|---|---|---|---|
| | 1 | 2 | 3 | 4 |
| A | • | • | | |
| B | • | | • | • |
| C | | • | • | |
| D | | • | | • |
| Weight | 3 | 5 | 4 | 6 |




$x_i = 1$ when we select set $i$

# Weighted Decision Diagrams

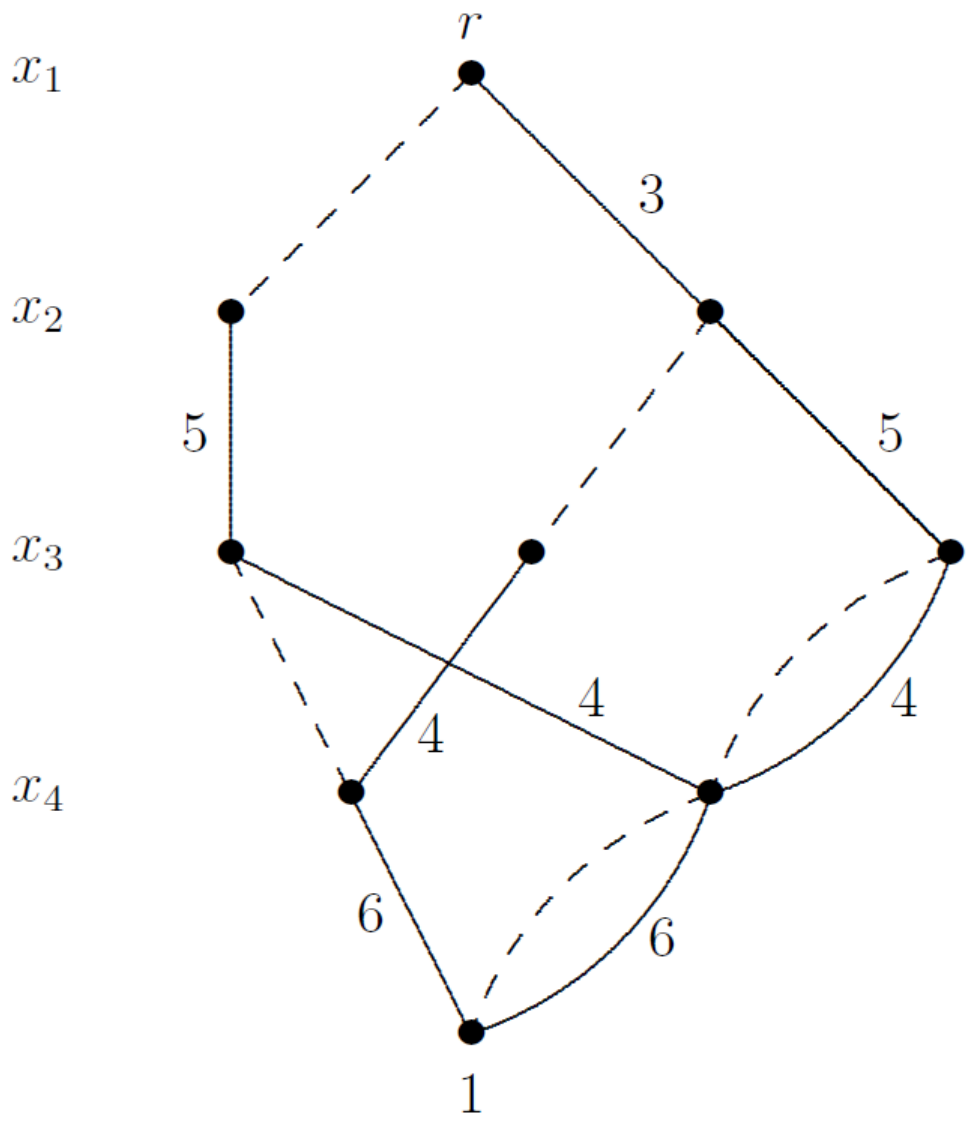
**Separable cost function**

Just label arcs with weights.

Shortest path corresponds to an optimal solution.

| | Set $i$ | | | |
|---|---|---|---|---|
| | 1 | 2 | 3 | 4 |
| A | • | • | | |
| B | • | | • | • |
| C | | • | • | |
| D | | • | | • |
| Weight | 3 | 5 | 4 | 6 |

$x_i = 1$ when we select set $i$



# Weighted Decision Diagrams

- State-dependent costs in dynamic programming imply a **nonseparable** cost function:

$$f(x) = \sum_{i=1}^{n} c_i(s_i, x_i)$$

where $s_{i+1} = \varphi_i(s_i, x_i), \quad i = 1, \ldots, n$

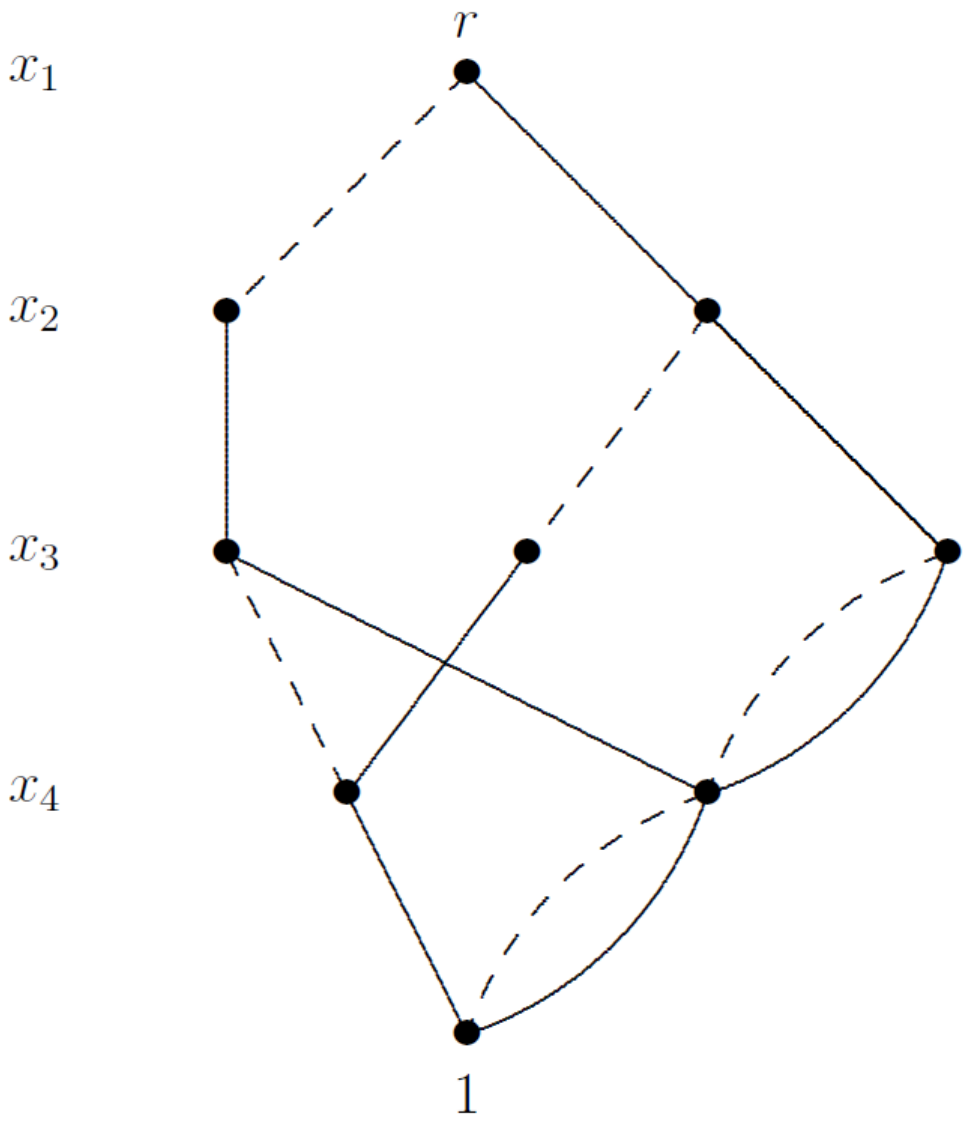
  - We need a theory of decision diagrams that deals with nonseparable costs.

# Weighted Decision Diagrams

**Nonseparable cost function**

Now what?

| $x$ | $f(x)$ |
|---|---|
| (0,1,0,1) | 6 |
| (0,1,1,0) | 7 |
| (0,1,1,1) | 8 |
| (1,0,1,1) | 5 |
| (1,1,0,0) | 6 |
| (1,1,0,1) | 8 |
| (1,1,1,0) | 7 |
| (1,1,1,1) | 9 |

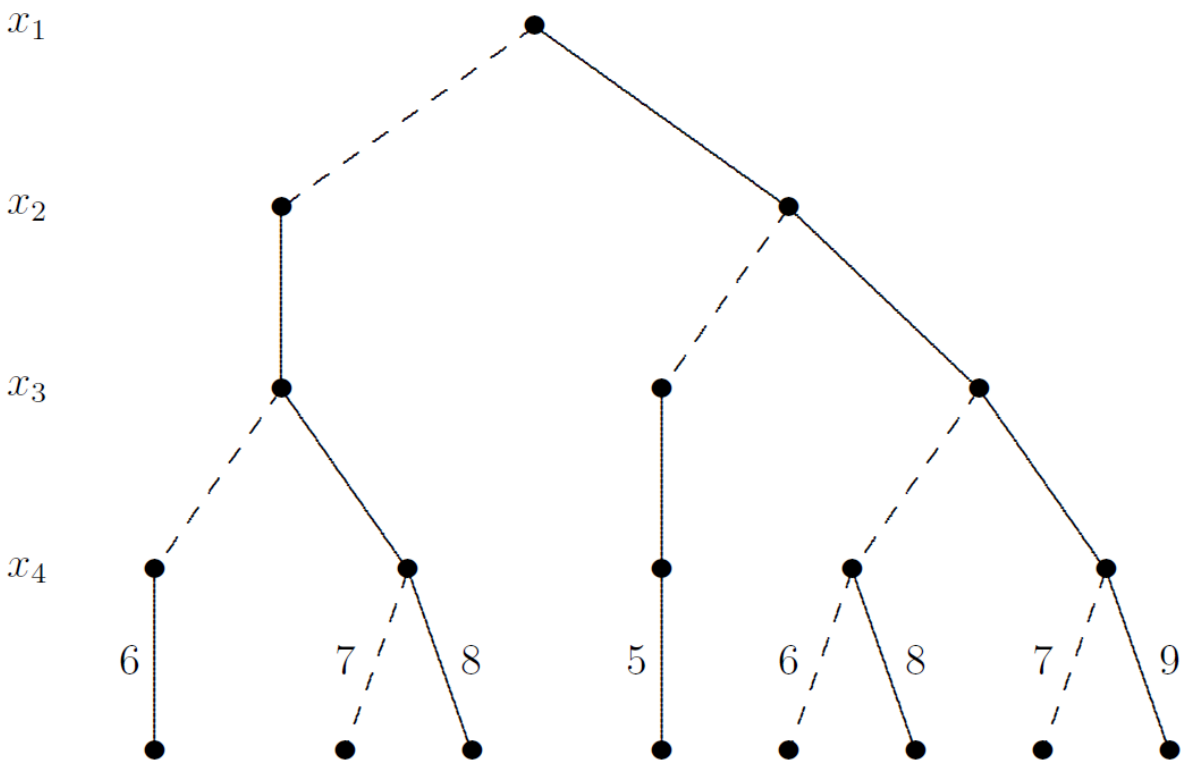


# Weighted Decision Diagrams

**Nonseparable cost function**

Put costs on leaves of branching tree.

| $x$ | $f(x)$ |
|---|---|
| (0,1,0,1) | 6 |
| (0,1,1,0) | 7 |
| (0,1,1,1) | 8 |
| (1,0,1,1) | 5 |
| (1,1,0,0) | 6 |
| (1,1,0,1) | 8 |
| (1,1,1,0) | 7 |
| (1,1,1,1) | 9 |

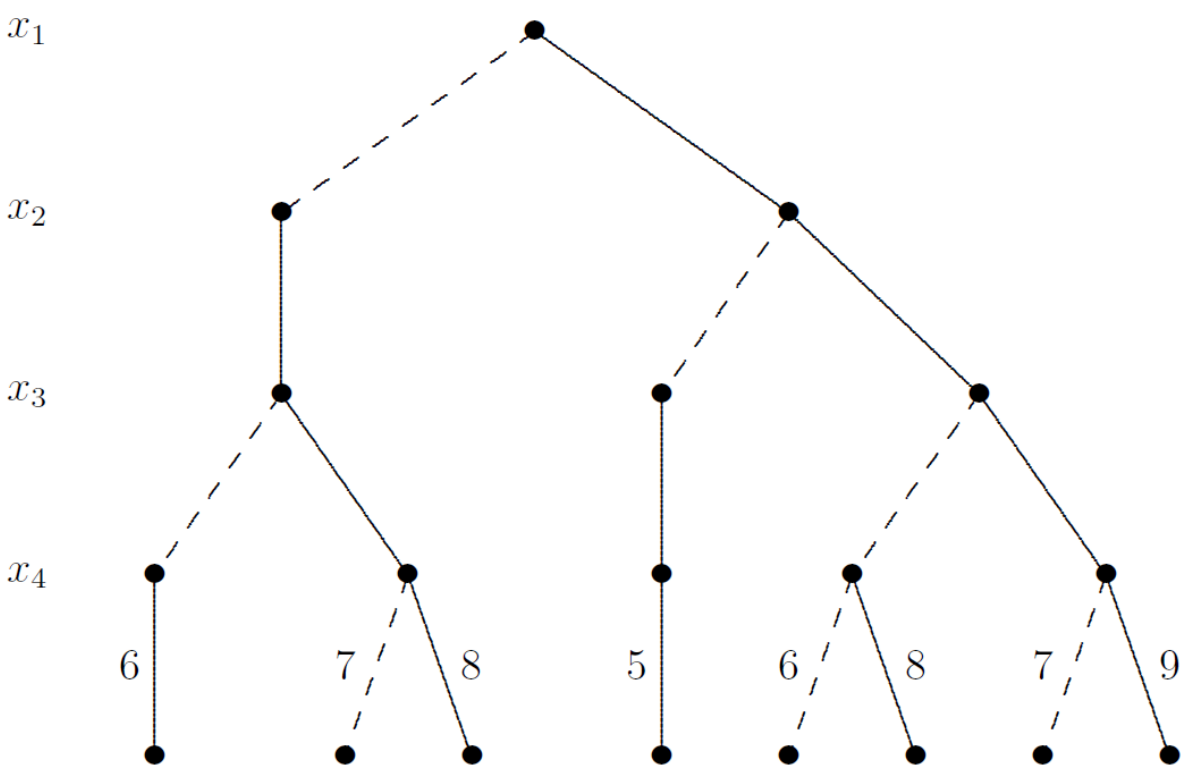


# Weighted Decision Diagrams

**Nonseparable cost function**

Put costs on leaves of branching tree.

But now we can't reduce the tree to an efficient decision diagram.



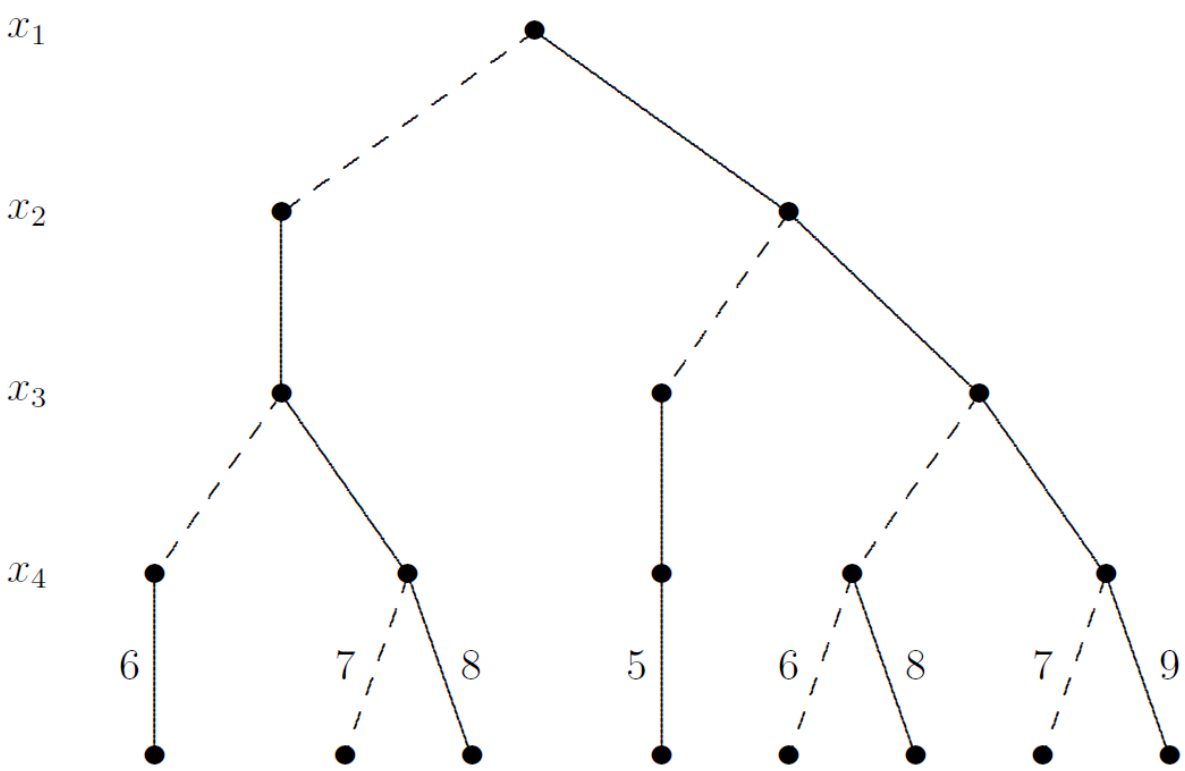
# Weighted Decision Diagrams

**Nonseparable cost function**

Put costs on leaves of branching tree.

But now we can't reduce the tree to an efficient decision diagram.

We will rearrange costs to obtain **canonical costs**.



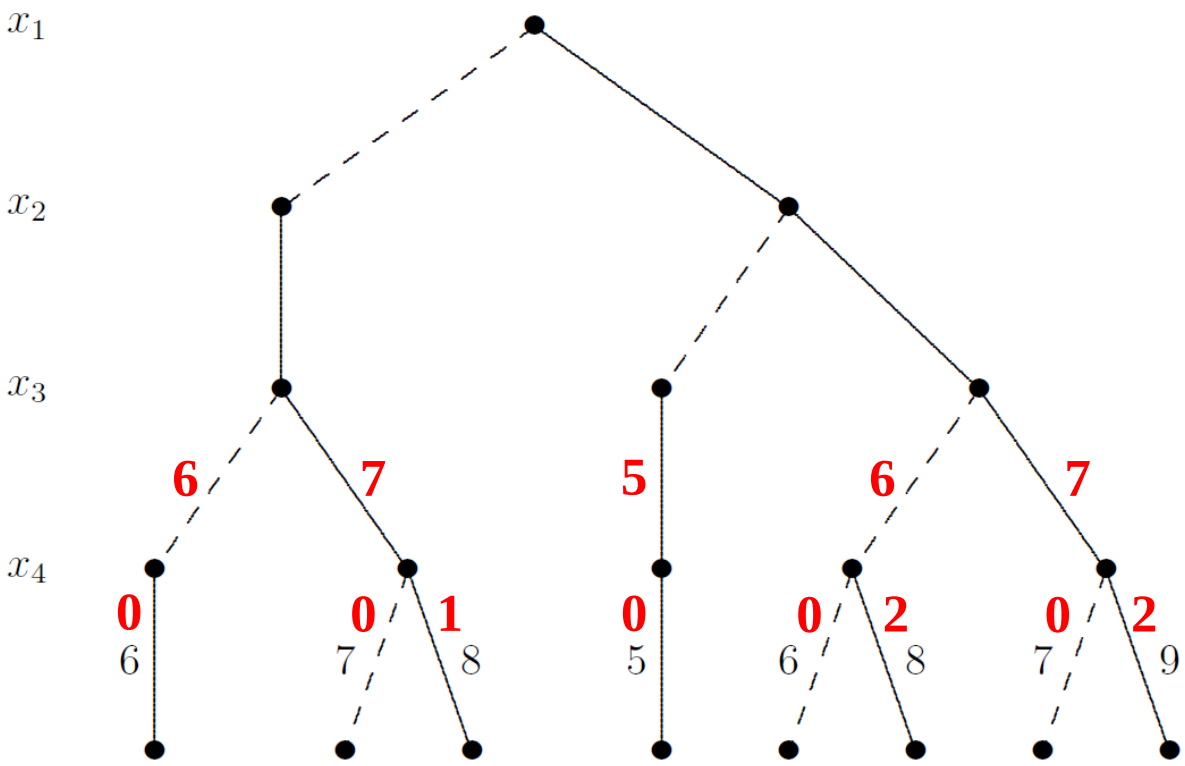
# Weighted Decision Diagrams

**Nonseparable cost function**

Put costs on leaves of branching tree.

But now we can't reduce the tree to an efficient decision diagram.

We will rearrange costs to obtain **canonical costs**.



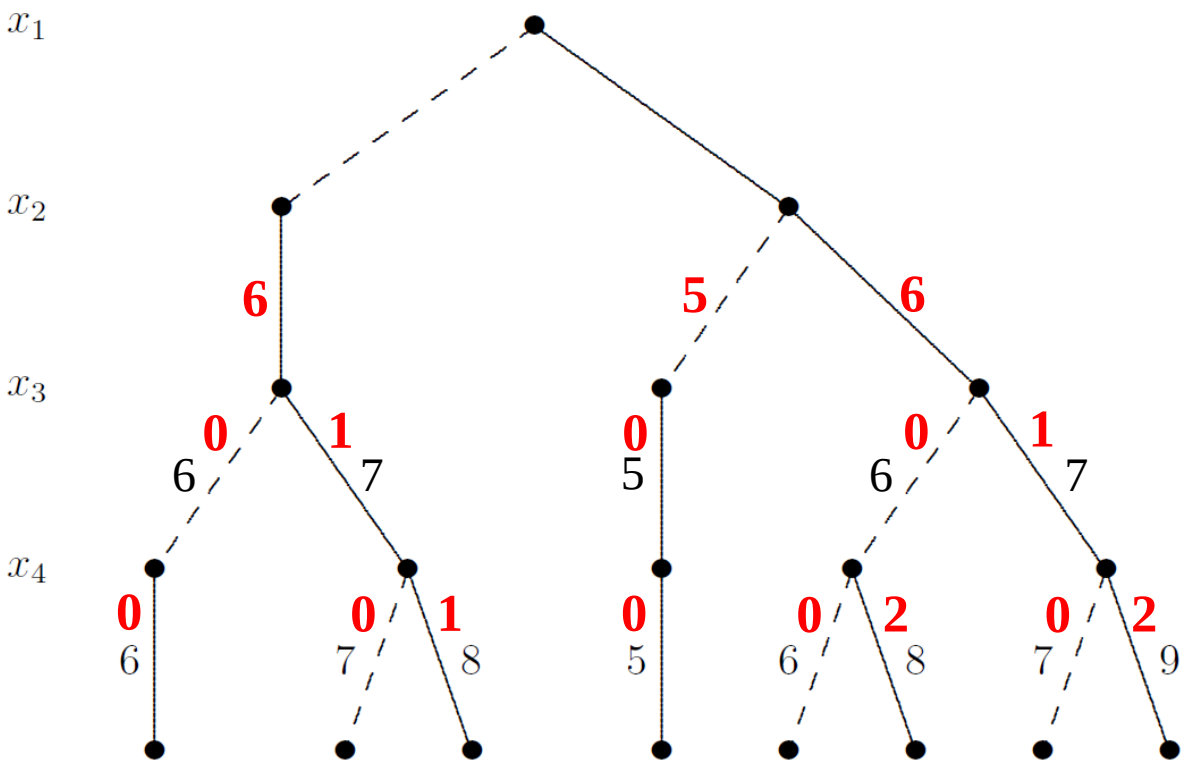
# Weighted Decision Diagrams

**Nonseparable cost function**

Put costs on leaves of branching tree.

But now we can't reduce the tree to an efficient decision diagram.

We will rearrange costs to obtain **canonical costs**.



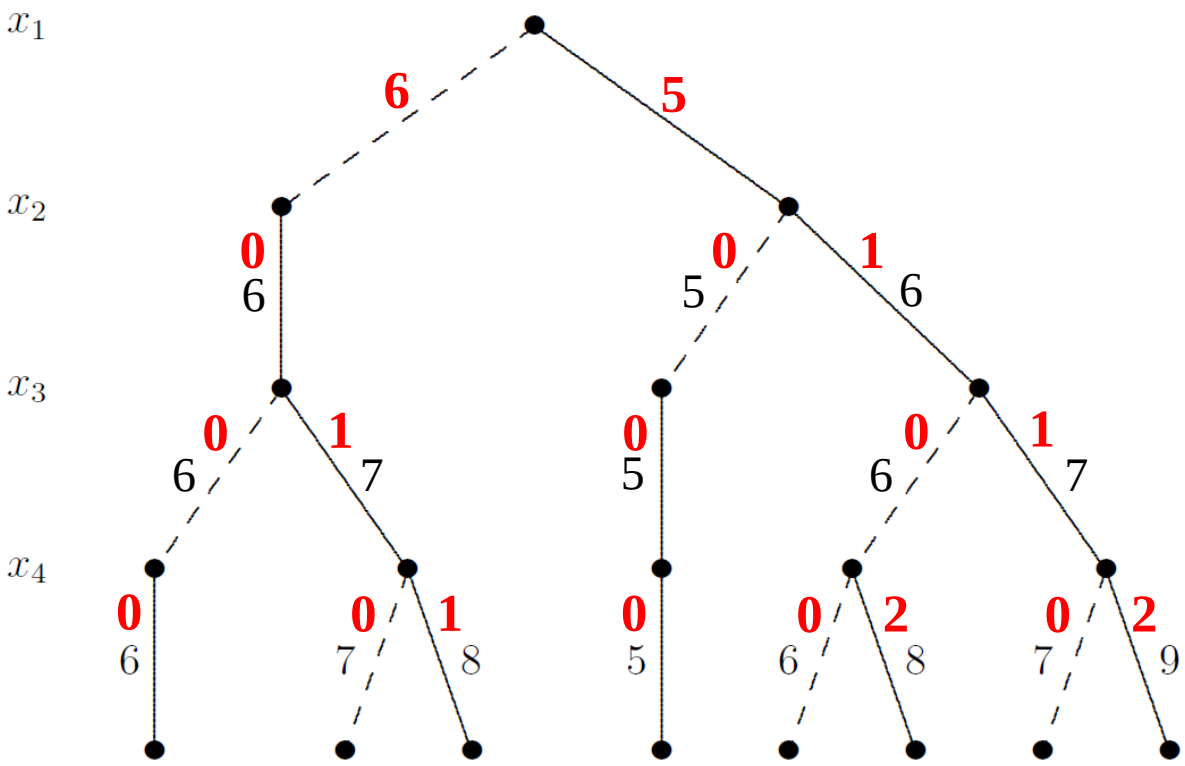
# Weighted Decision Diagrams

**Nonseparable cost function**

Put costs on leaves of branching tree.

But now we can't reduce the tree to an efficient decision diagram.

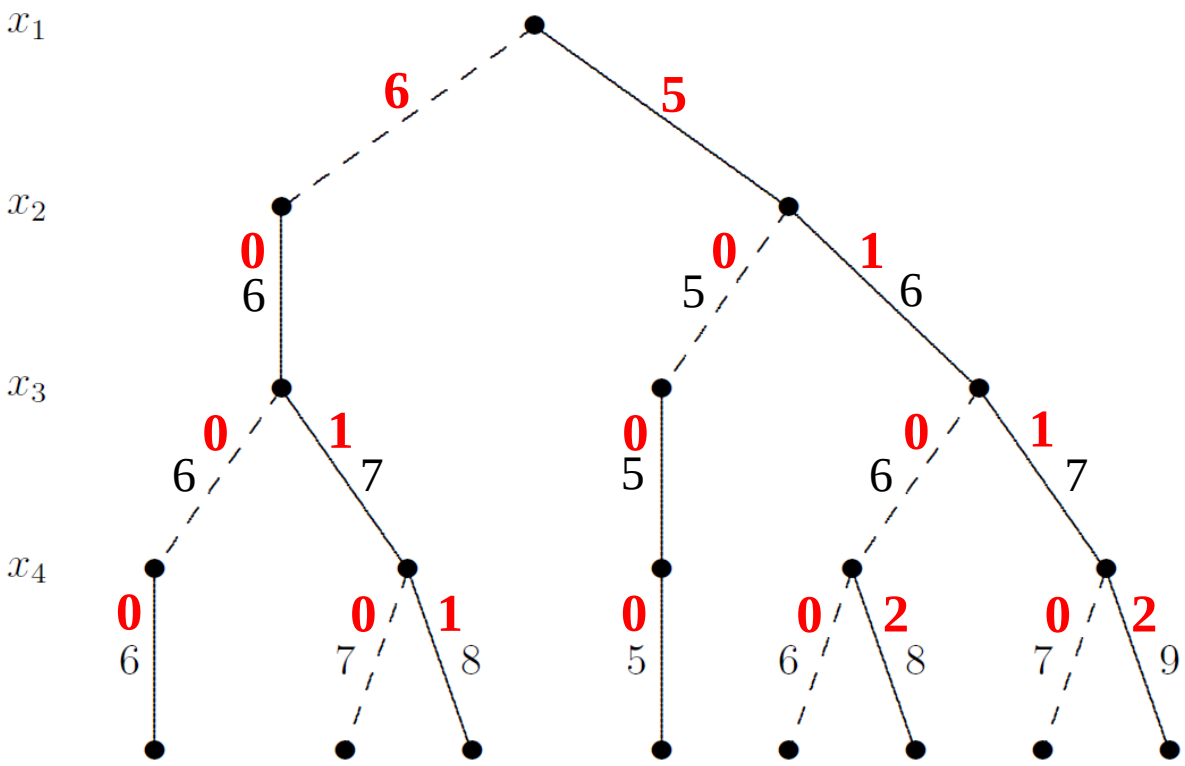
We will rearrange costs to obtain **canonical costs**.



# Weighted Decision Diagrams

**Nonseparable cost function**

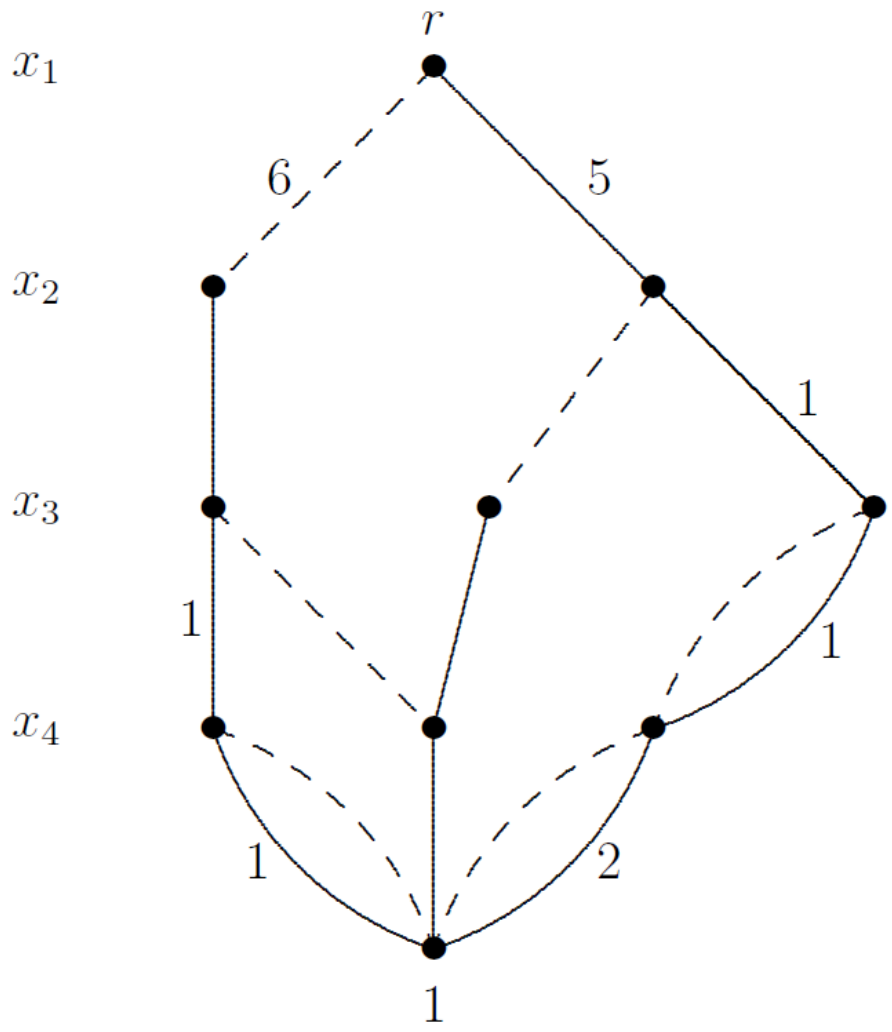
Now the tree can be reduced.



# Weighted Decision Diagrams

**Nonseparable cost function**

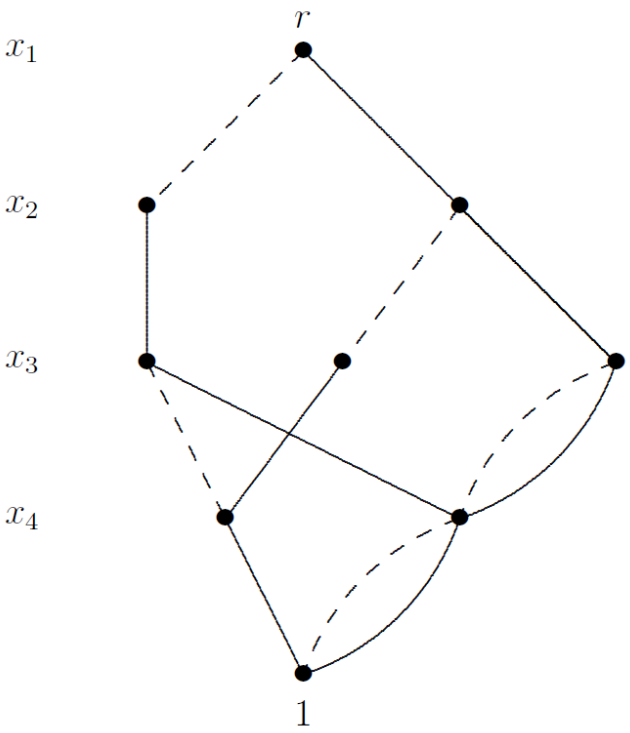
Now the tree can
be reduced.

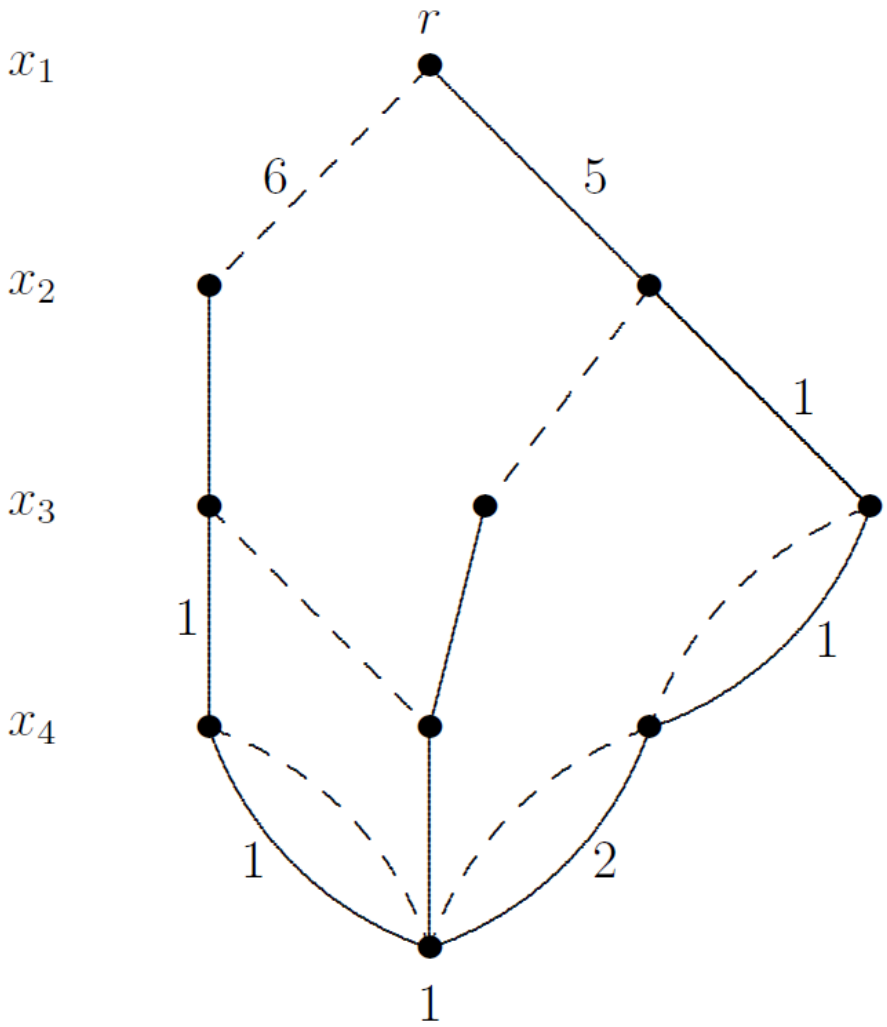


# Weighted Decision Diagrams

**Nonseparable cost function**

Note that DD is larger than reduced unweighted DD, but still compact.



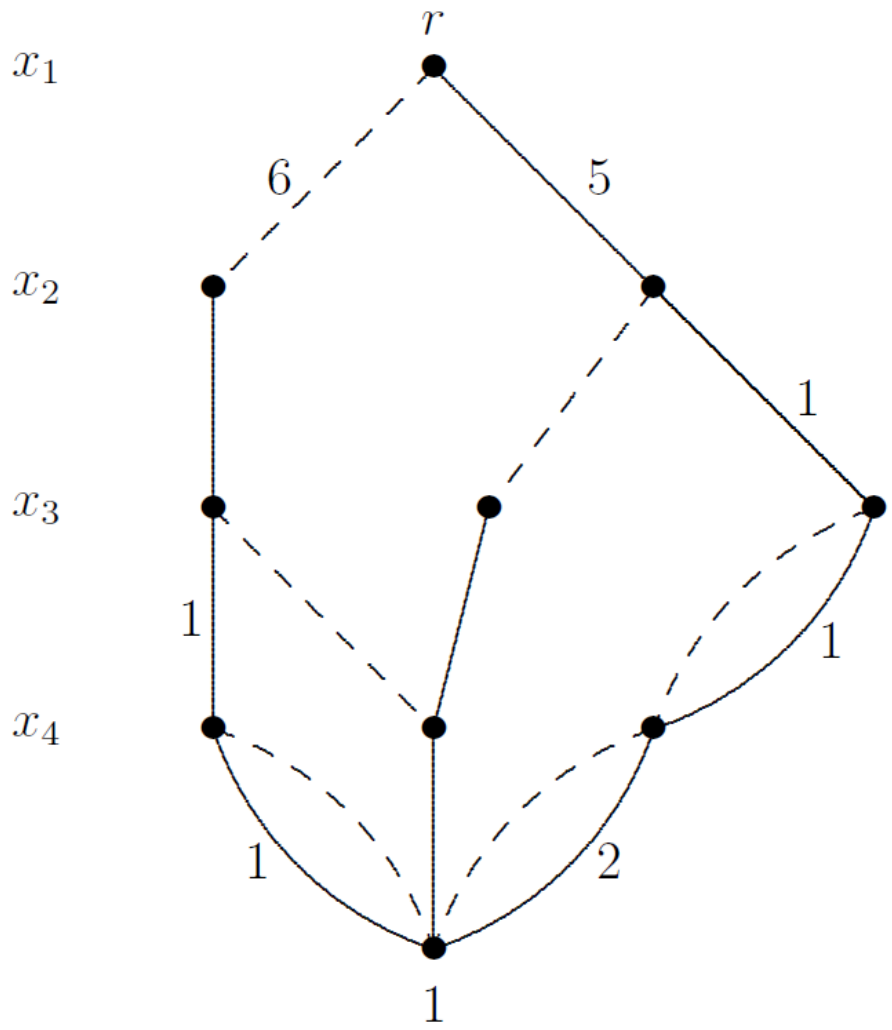



# Weighted Decision Diagrams

**Nonseparable cost function**

We can represent any discrete optimization problem with such a decision diagram

even if the costs are nonseparable.



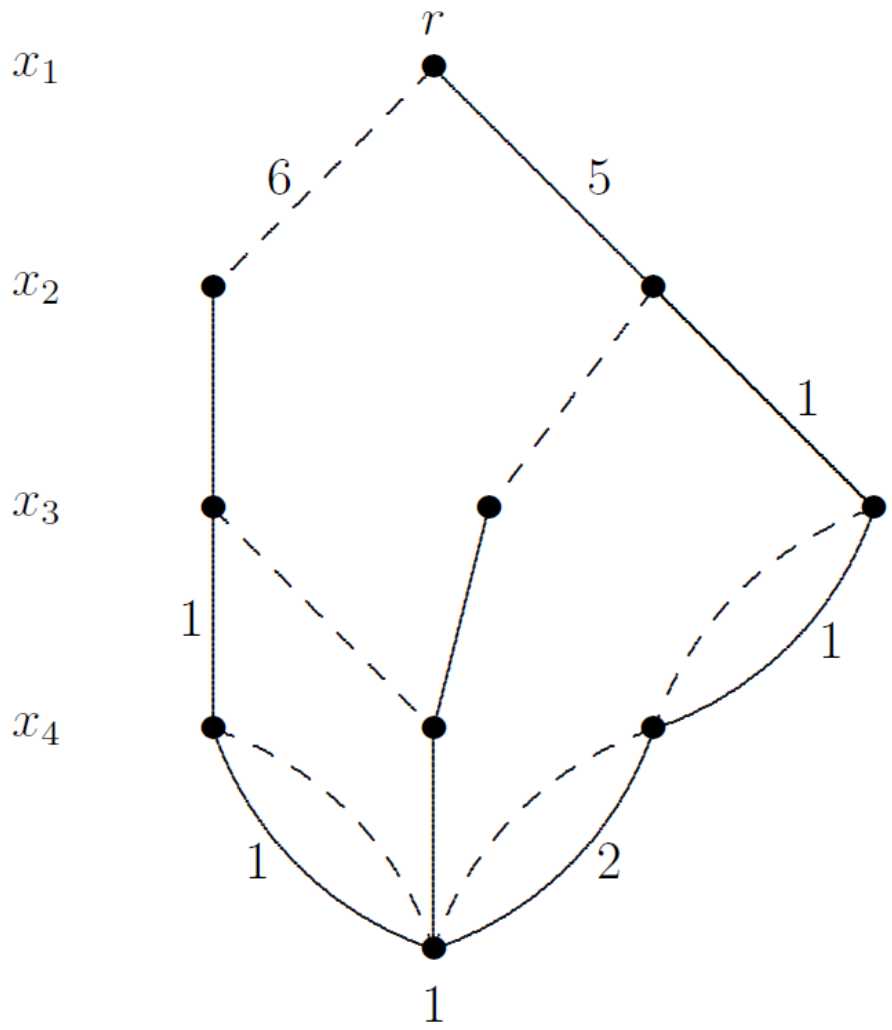
# Weighted Decision Diagrams

**Nonseparable cost function**

We know that without weights, there is a **unique** reduced decision diagram for a given variable ordering.

Is this true for decision diagrams with **canonical** weights?

**Yes.**



# Weighted Decision Diagrams

**Definition.** Costs on a decision diagram are **canonical** if for every node in layer $i$, the costs $c_{ij}$ leaving that node satisfy

$$\min_j \{c_{ij}\} = \alpha_i$$

for fixed $\alpha_i$ (e.g., 0).

# Weighted Decision Diagrams

**Definition.** Costs on a decision diagram are **canonical** if for every node in layer $i$, the costs $c_{ij}$ leaving that node satisfy

$$\min_j \{c_{ij}\} = \alpha_i$$

for fixed $\alpha_i$ (e.g., 0).

**Theorem.** Any given discrete optimization problem is **uniquely** represented by a weighted decision diagram with canonical costs, for a given variable ordering.

# Weighted Decision Diagrams

**Definition.** Costs on a decision diagram are **canonical** if for every node in layer $i$, the costs $c_{ij}$ leaving that node satisfy

$$\min_j \{c_{ij}\} = \alpha_i$$

for fixed $\alpha_i$ (e.g., 0).

**Theorem.** Any given discrete optimization problem is **uniquely** represented by a weighted decision diagram with canonical costs, for a given variable ordering.

- Similar result proved for Affine Algebraic Decision Diagrams (AADDs) by Sanner and McAllester (IJCAI 2005).
  - Definition of **canonical** is somewhat different.

# Weighted Decision Diagrams

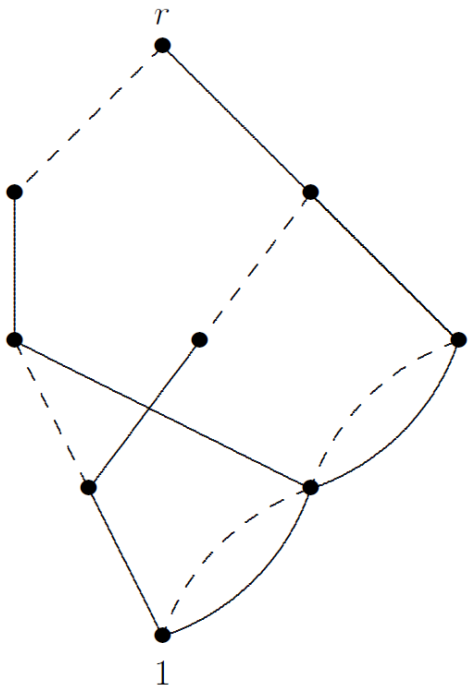
- Converting to canonical costs does not destroy the benefits of separability.

**Definition.** A decision diagram is **separable** when arc costs represent terms of a separable cost function.

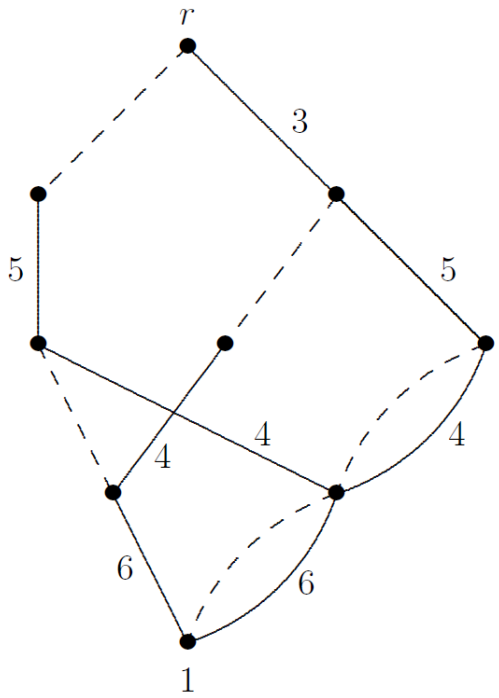
**Theorem.** A separable decision diagram that is reduced when costs are ignored is also reduced when costs are converted to canonical costs.

# Weighted Decision Diagrams

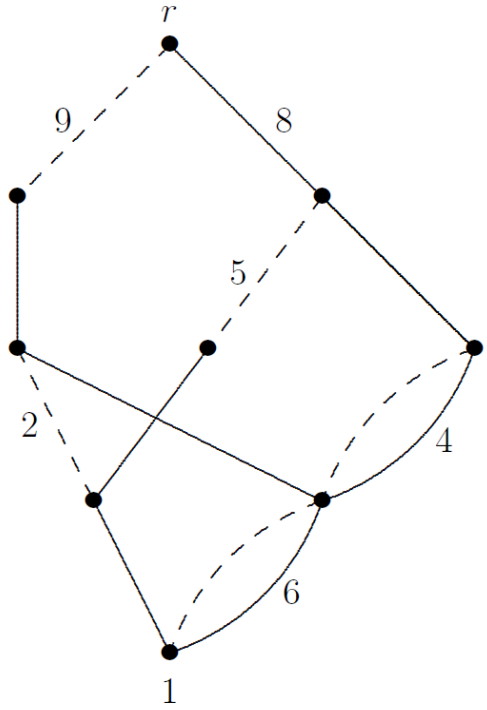
**Example**




Reduced unweighted DD

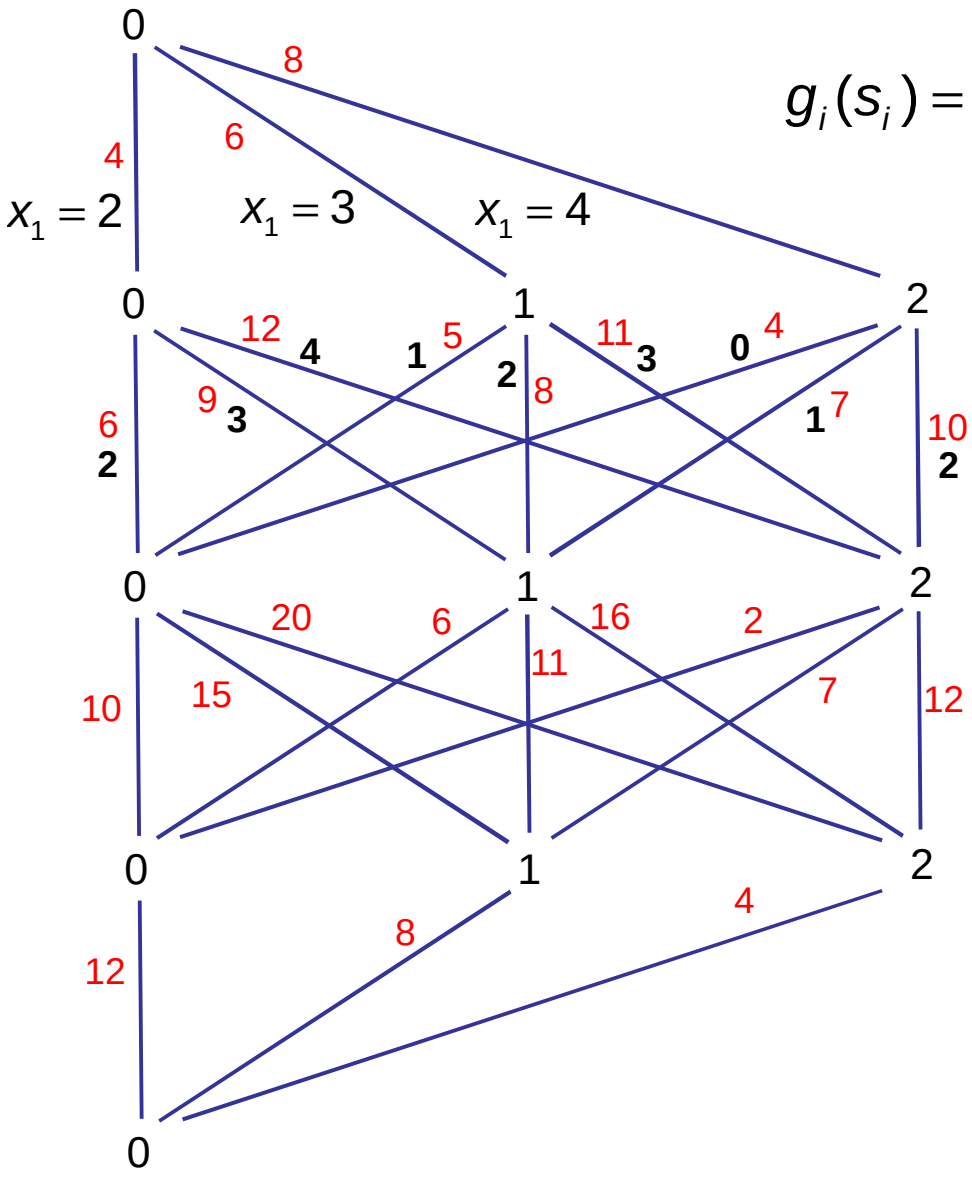



Add separable costs




Reduced weighted DD with canonical costs has same shape

# Application to Inventory Problem



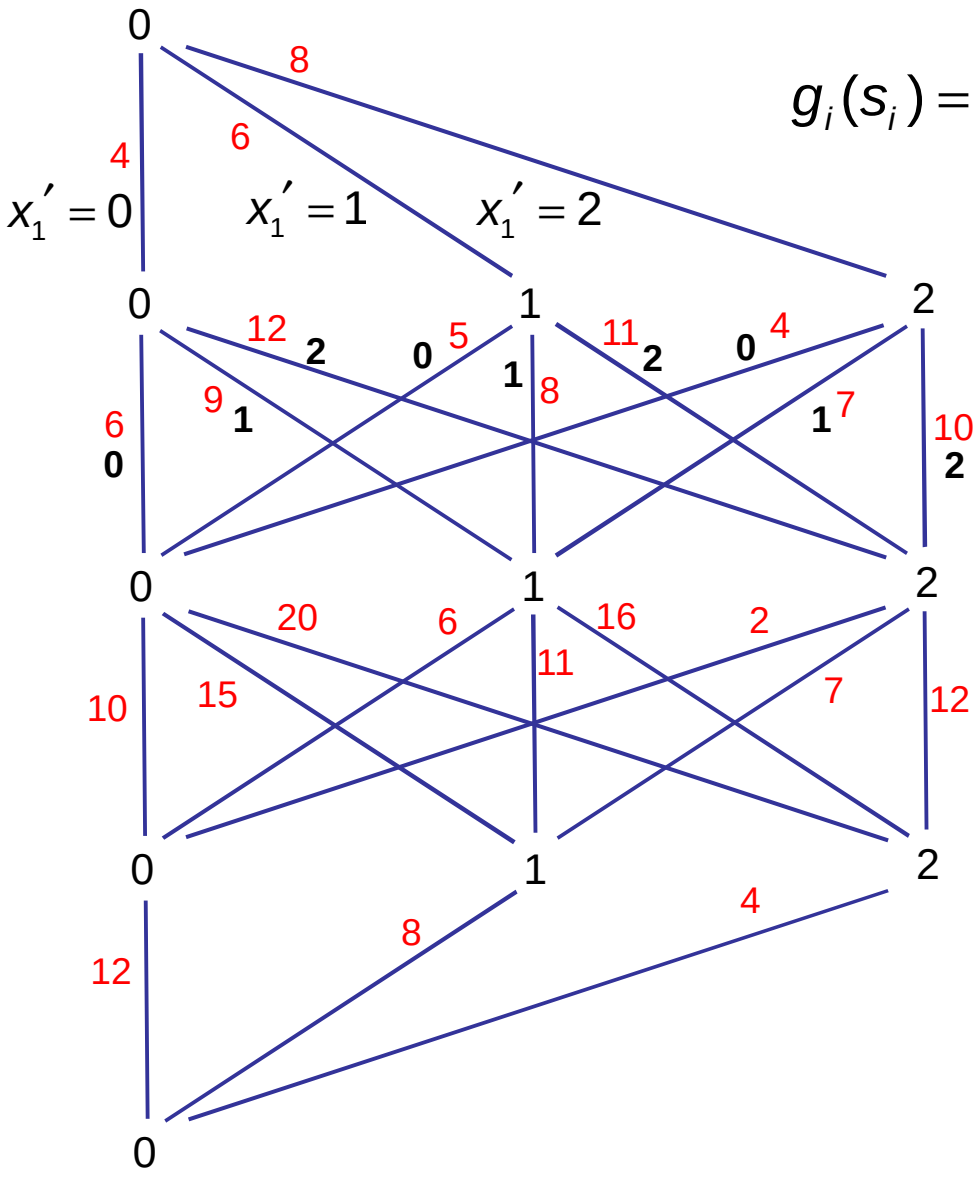

$$g_i(s_i) = \min_{x_i}\{h_i s_i + c_i x_i + g_{i+1}(s_i + x_i - d_i)\}$$

To equalize controls, let

$$x_i' = s_i + x_i - d_i$$

Be the stock level in next period.

# Application to Inventory Problem



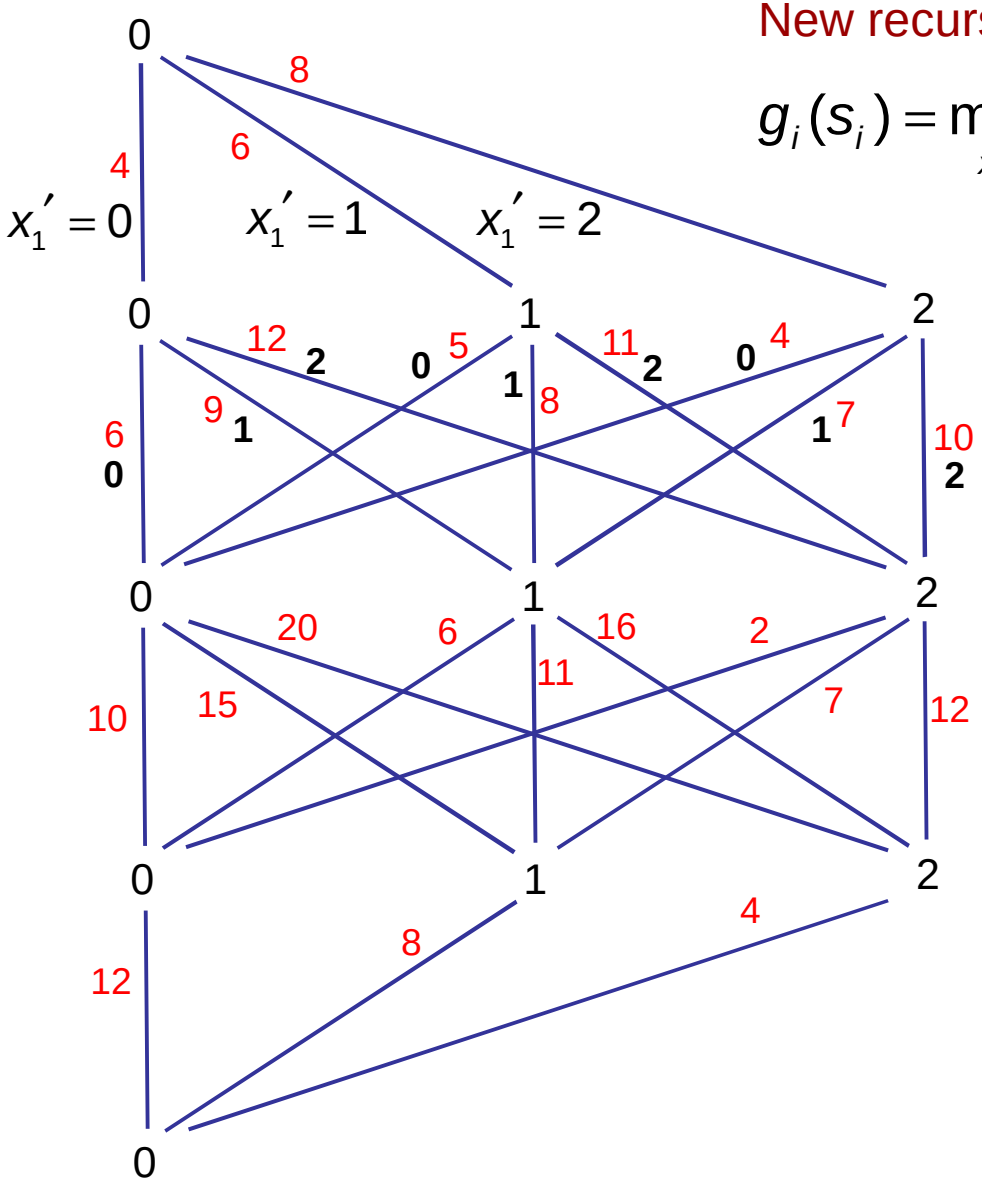

$$g_i(s_i) = \min_{x_i}\{h_i s_i + c_i x_i + g_{i+1}(s_i + x_i - d_i)\}$$

To equalize controls, let

$$x_i' = s_i + x_i - d_i$$

Be the stock level in next period.

# Application to Inventory Problem




New recursion:

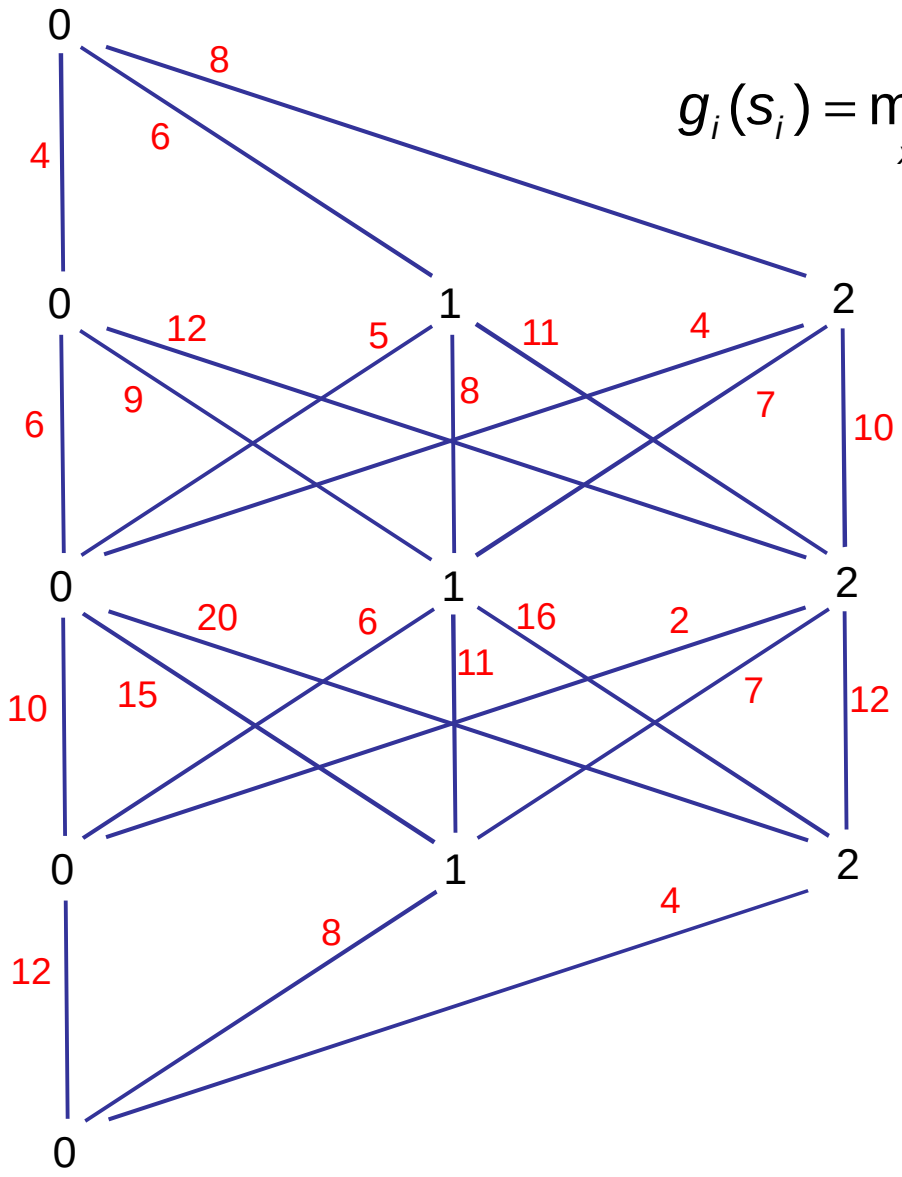
$$g_i(s_i) = \min_{x_i'} \left\{ h_i s_i + c_i(x_i' - s_i + d_i) + g_{i+1}(x_i') \right\}$$

To equalize controls, let

$$x_i' = s_i + x_i - d_i$$

Be the stock level in next period.

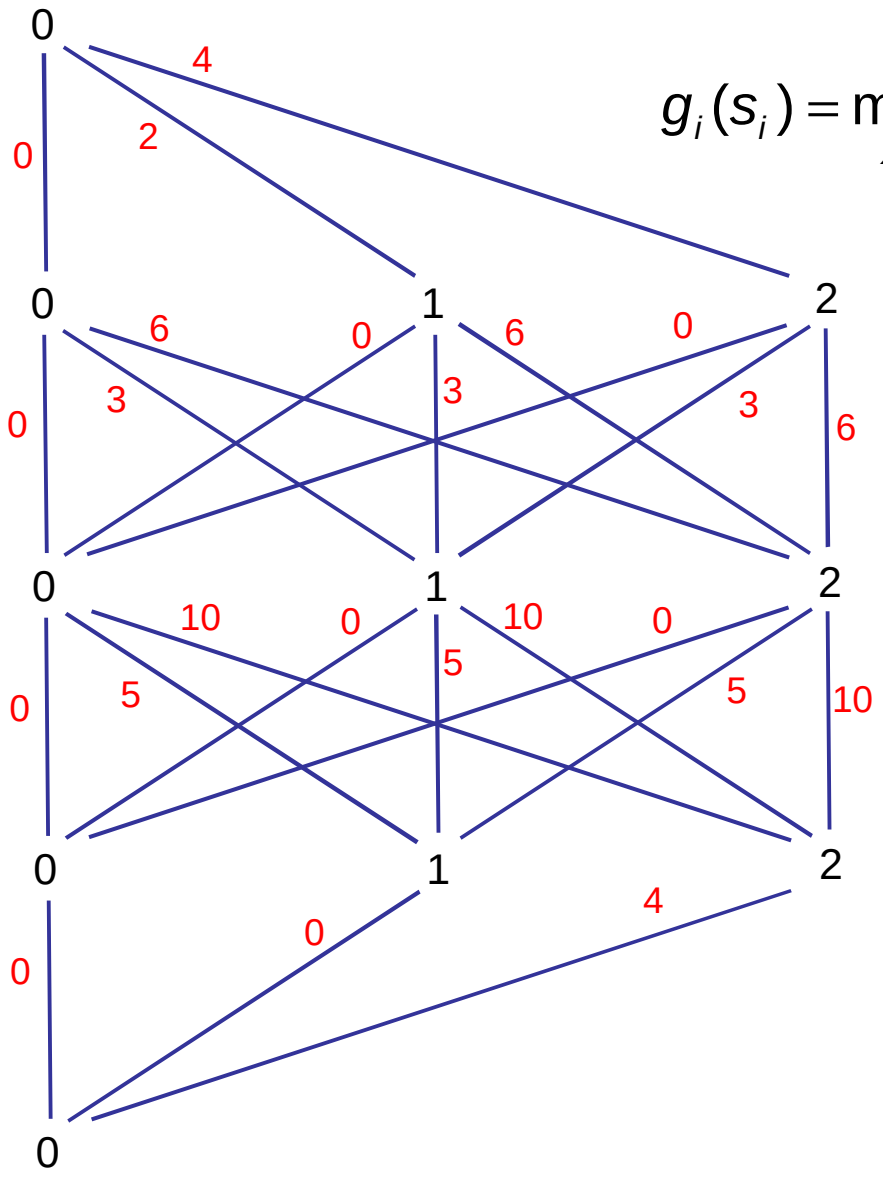
# Application to Inventory Problem




$$g_i(s_i) = \min_{x_i'} \left\{ h_i s_i + c_i(x_i' - s_i + d_i) + g_{i+1}(x_i') \right\}$$

To obtain canonical costs, subtract $c_i(m - s_i) + h_i s_i$ from cost on each arc $(s_i, s_{i+1})$.

# Application to Inventory Problem

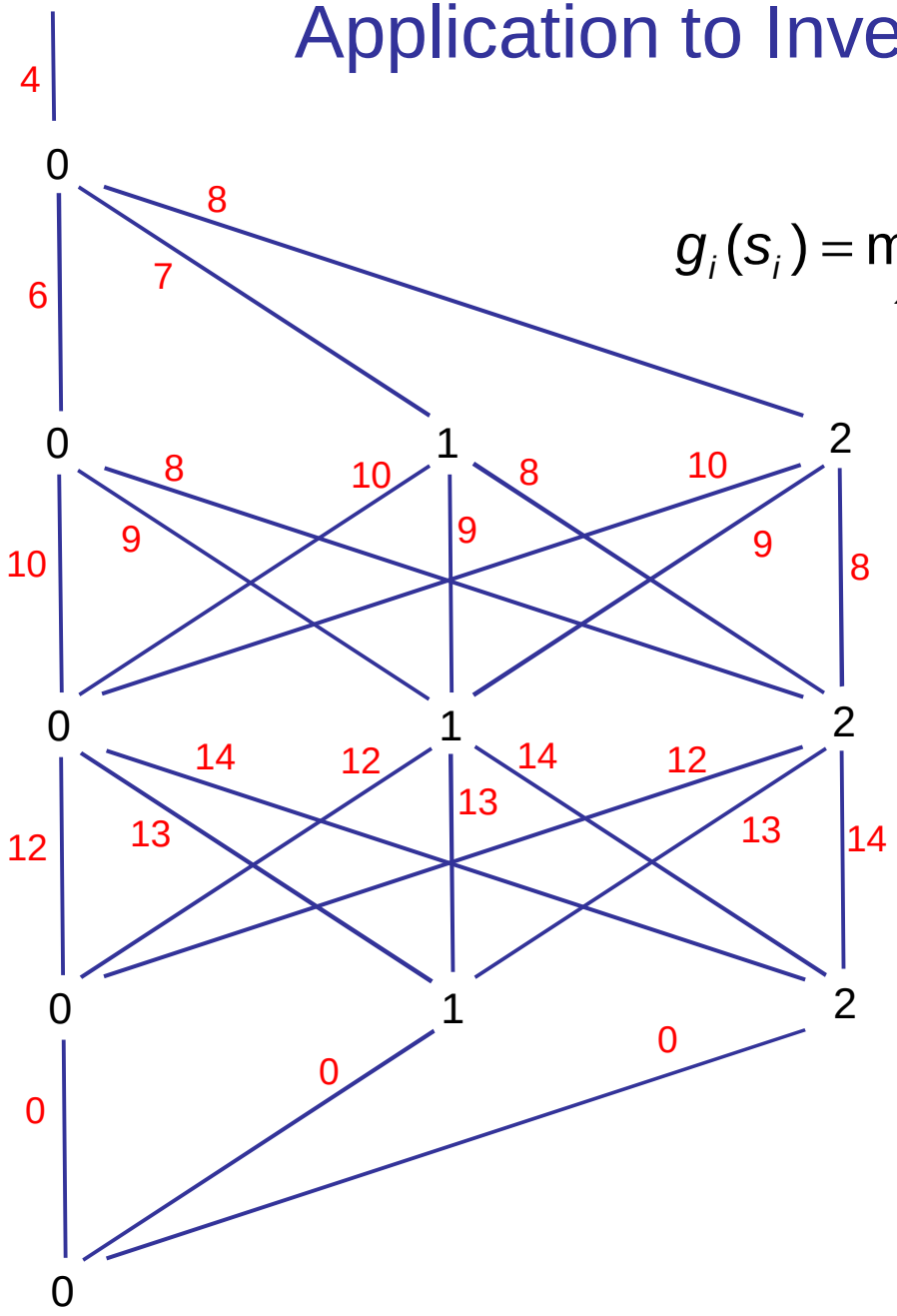



$$g_i(s_i) = \min_{x_i'} \left\{ h_i s_i + c_i(x_i' - s_i + d_i) + g_{i+1}(x_i') \right\}$$

To obtain canonical costs, subtract $c_i(m - s_i) + h_i s_i$ from cost on each arc $(s_i, s_{i+1})$.

Add these offsets to incoming arcs.

# Application to Inventory Problem

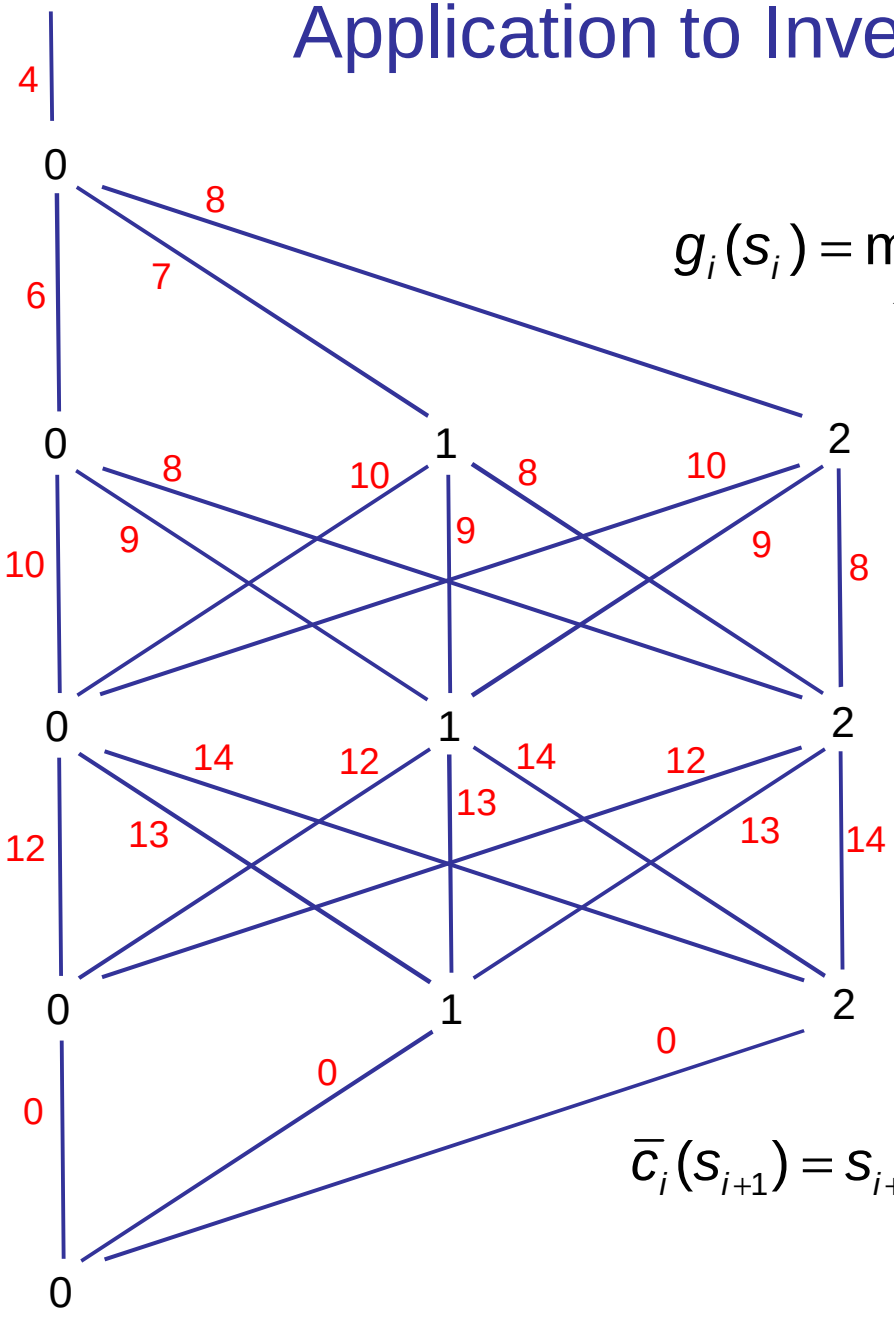



$$g_i(s_i) = \min_{x_i'} \left\{ h_i s_i + c_i(x_i' - s_i + d_i) + g_{i+1}(x_i') \right\}$$

To obtain canonical costs, subtract $c_i(m - s_i) + h_i s_i$ from cost on each arc $(s_i, s_{i+1})$.

Add these offsets to incoming arcs.

# Application to Inventory Problem




$$g_i(s_i) = \min_{x_i'} \left\{ h_i s_i + c_i(x_i' - s_i + d_i) + g_{i+1}(x_i') \right\}$$

To obtain canonical costs, subtract $c_i(m - s_i) + h_i s_i$ from cost on each arc $(s_i, s_{i+1})$.

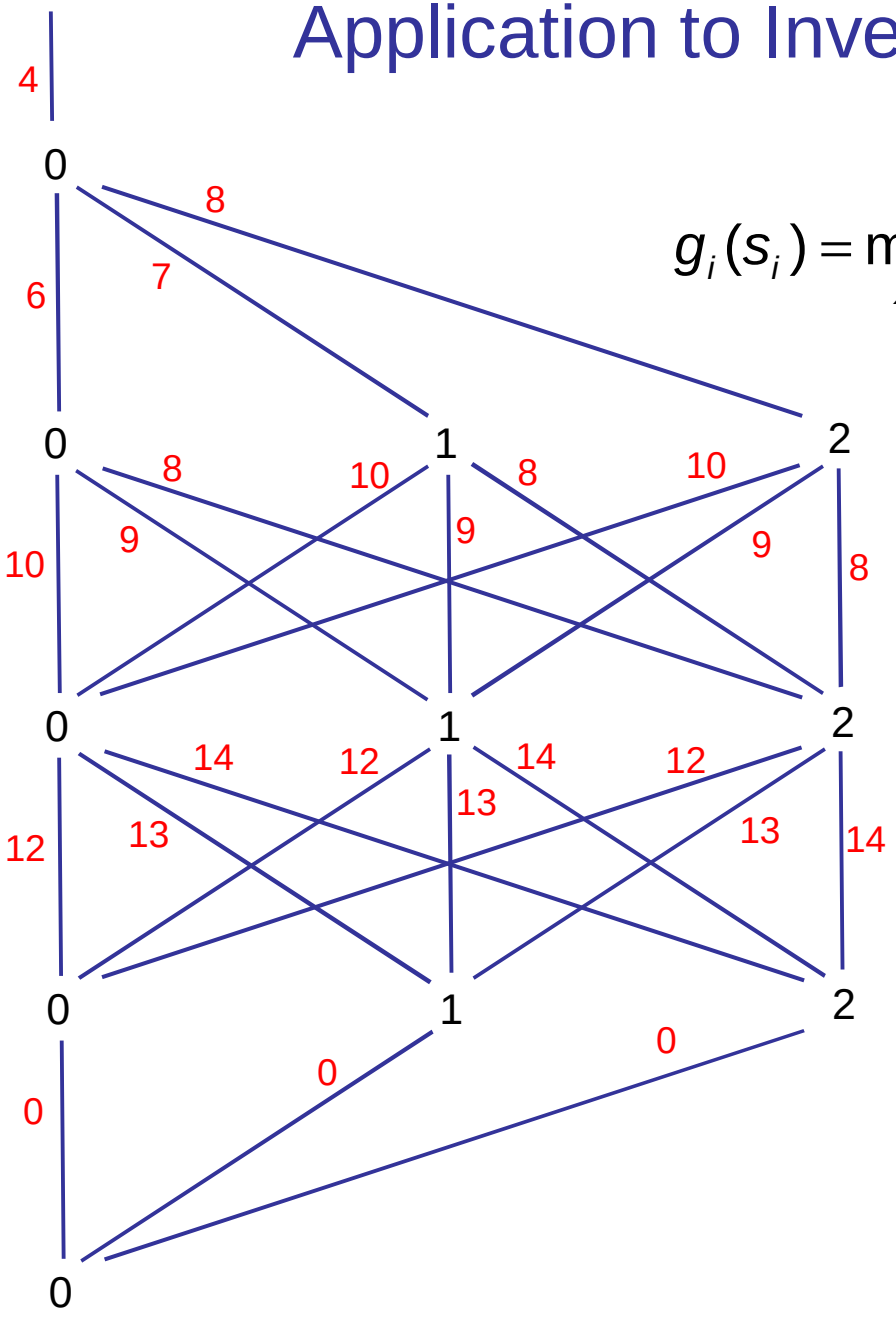
Add these offsets to incoming arcs.

Now outgoing arcs look alike.

And all arcs into state $s_i$ have the same cost

$$\bar{c}_i(s_{i+1}) = s_{i+1} h_{i+1} + c_i(d_i - s_{i+1} - m) + c_{i+1}(m - s_{i+1})$$

# Application to Inventory Problem

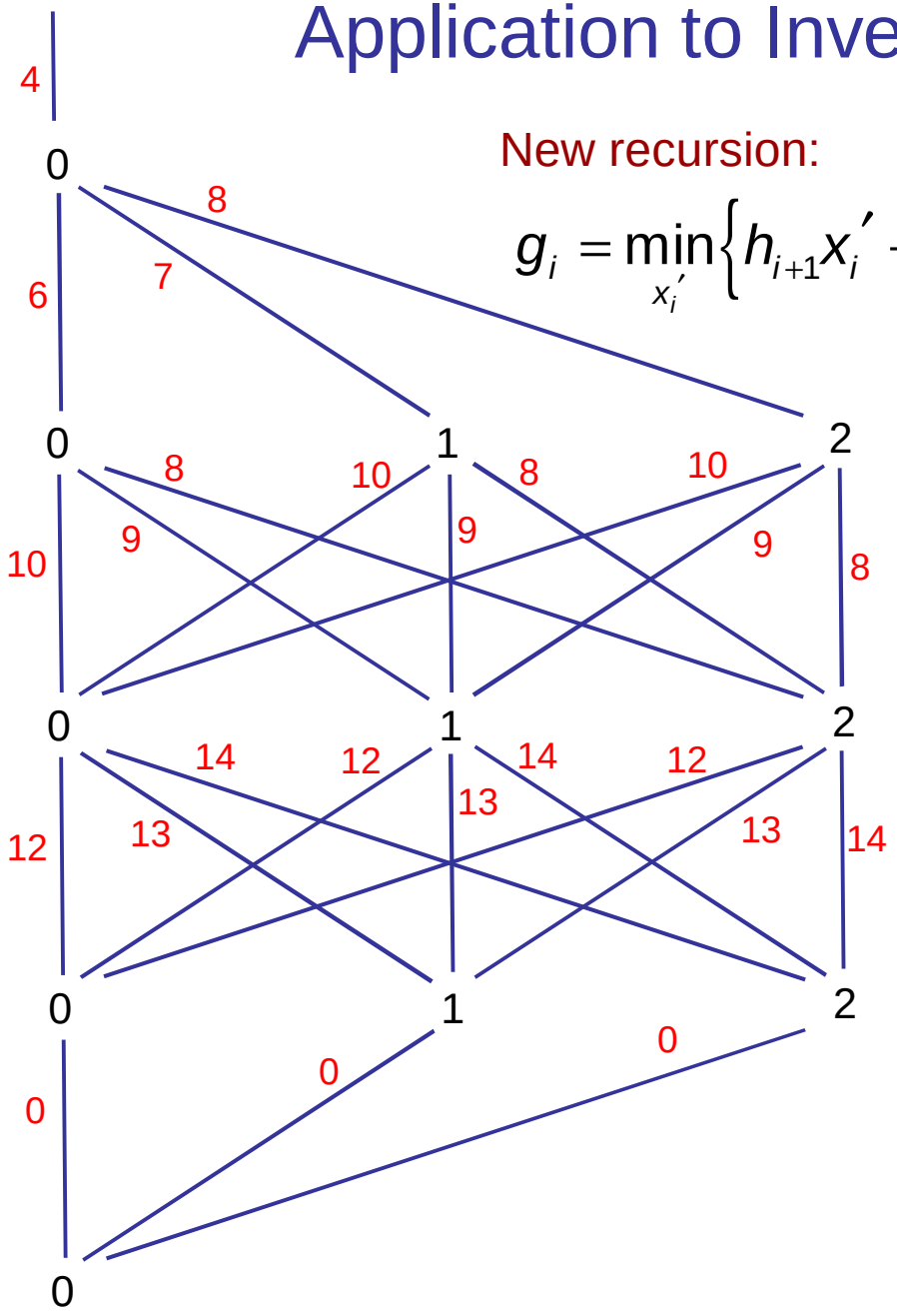



$$g_i(s_i) = \min_{x_i'} \left\{ h_i s_i + c_i(x_i' - s_i + d_i) + g_{i+1}(x_i') \right\}$$

These are canonical costs with

$$\alpha_i = \min_{s_{i+1}} \left\{ \bar{c}_i(s_{i+1}) \right\}$$

# Application to Inventory Problem



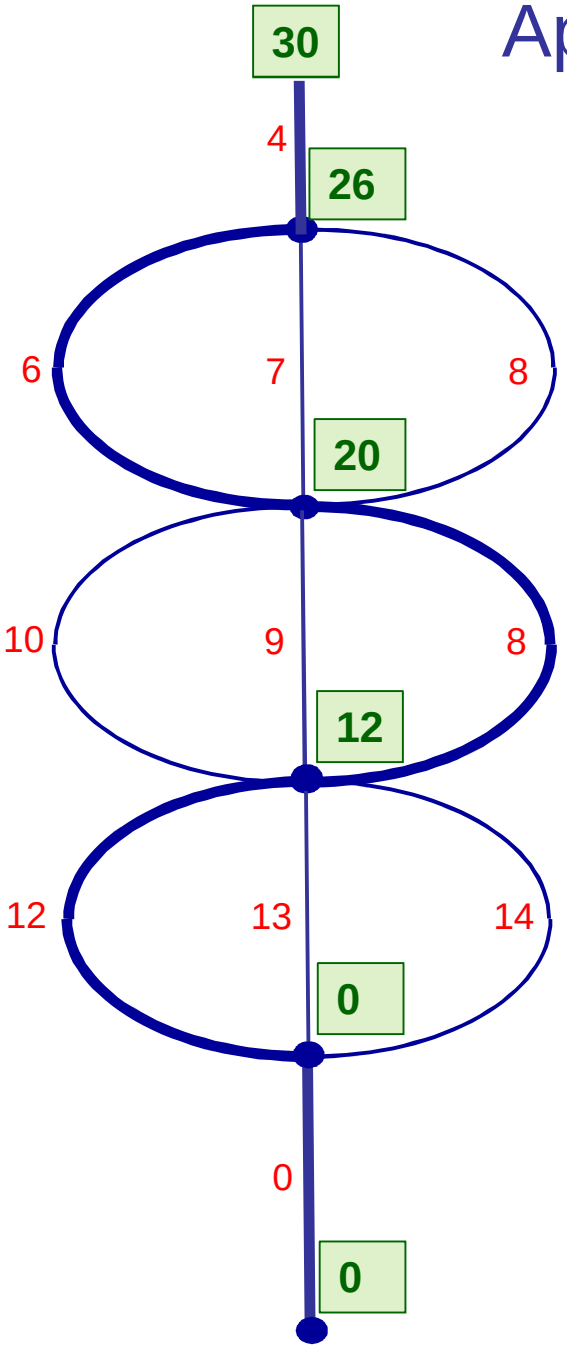

New recursion:

$$g_i = \min_{x_i'} \left\{ h_{i+1} x_i' + c_i(x_i' - m + d_i) + c_{i+1}(m - x_i') + g_{i+1} \right\}$$

These are canonical costs with

$$\alpha_i = \min_{s_{i+1}} \left\{ \bar{c}_i(s_{i+1}) \right\}$$

# Application to Inventory Problem

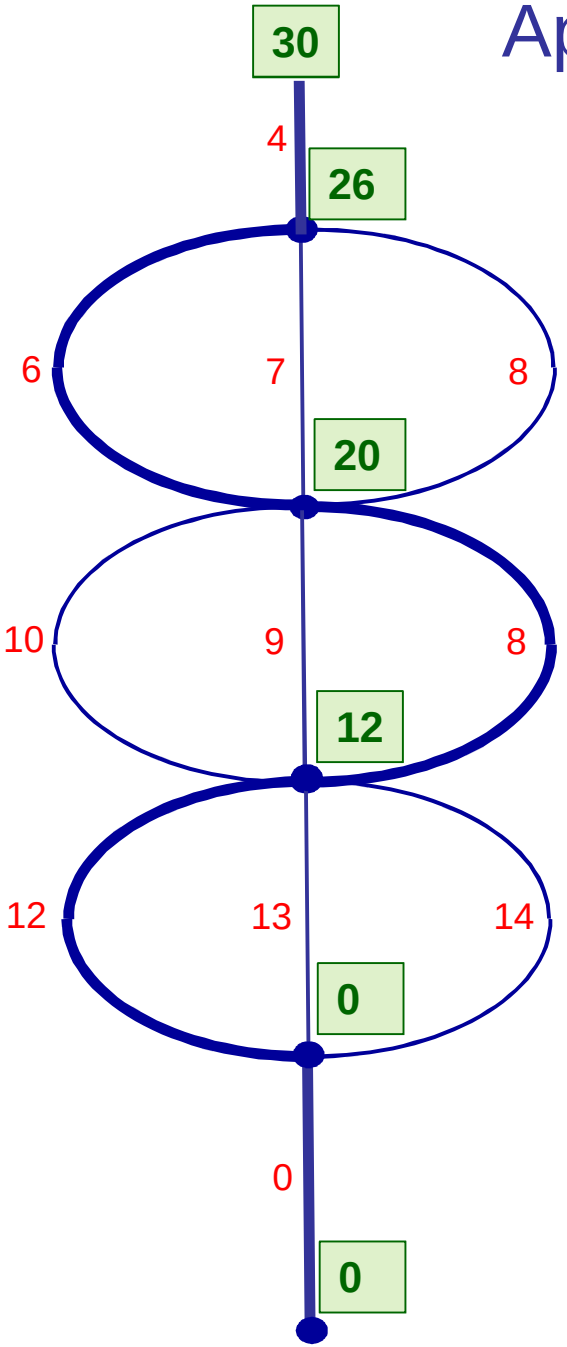



New recursion:

$$g_i = \min_{x_i'} \left\{ h_{i+1} x_i' + c_i (x_i' - m + d_i) + c_{i+1}(m - x_i') + g_{i+1} \right\}$$

Now there is only one state per period.

# Application to Inventory Problem




New recursion:

$$g_i = \min_{x_i'} \left\{ h_{i+1} x_i' + c_i (x_i' - m + d_i) + c_{i+1}(m - x_i') + g_{i+1} \right\}$$

Now there is only one state per period.

Note that computational tests are not necessary.

We immediately see the speedup from the reduction in the state space.

# Future Research

- DP model simplification
  - Go through the classical DP models and see under what conditions they can be simplified.

# Future Research

- DP model simplification
  - Go through the classical DP models and see under what conditions they can be simplified.
- New approach to approximate DP
  - Approximate DP merges similar states to reduce state space.
    - Approximation scheme is normally static (fixed in advance).

# Future Research

- DP model simplification
  - Go through the classical DP models and see under what conditions they can be simplified.
- New approach to approximate DP
  - Approximate DP merges similar states to reduce state space.
    - Approximation scheme is normally static (fixed in advance).
  - Relaxation techniques for decision diagrams do the same.
    - Choose which nodes to merge as the diagram is built.

# Future Research

- DP model simplification
  - Go through the classical DP models and see under what conditions they can be simplified.
- New approach to approximate DP
  - Approximate DP merges similar states to reduce state space.
    - Approximation scheme is normally static (fixed in advance).
  - Relaxation techniques for decision diagrams do the same.
    - Choose which nodes to merge as the diagram is built.
  - Apply these techniques to state transition graph.

# Future Research

- DP model simplification
  - Go through the classical DP models and see under what conditions they can be simplified.
- New approach to approximate DP
  - Approximate DP merges similar states to reduce state space.
    - Approximation scheme is normally static (fixed in advance).
  - Relaxation techniques for decision diagrams do the same.
    - Choose which nodes to merge as the diagram is built.
  - Apply these techniques to state transition graph.
  - Result is a **dynamic dynamic programming**
    - DP with dynamic state approximation based on information in the decision diagram.