

Computer Algorithms

Unit-3

Dynamic Programming

The General Method

- An algorithm design method that can be used
- When the solution to a problem can be viewed
- as the result of a **sequence of decisions**.
- Step wise decisions can be made for the problem like – knapsack problem, job sequencing with deadline, optimal merge pattern etc.
- Which can be solved using Greedy Approach

The General Method

- For some problems it is not possible to make a sequence of stepwise decisions, which will give optimal decision sequence.
- Solution to such problems is – to **try all possible decision sequences**
- Enumerate all decision sequences and then pick out best.
- But the time and space requirement may be prohibitive.

The General Method

- Dynamic Programming drastically reduces the amount of time and space required.
- It avoids the enumeration of some decision sequences , that cannot possibly be optimal

Difference between Greedy Method and Dynamic Programming

- In Greedy Method, only one decision sequence is even generated.
- In Dynamic Programming, many decision sequences can be generated
- But sequences containing suboptimal subsequences can not be optimal
- And so will not be generated

Multistage Graph

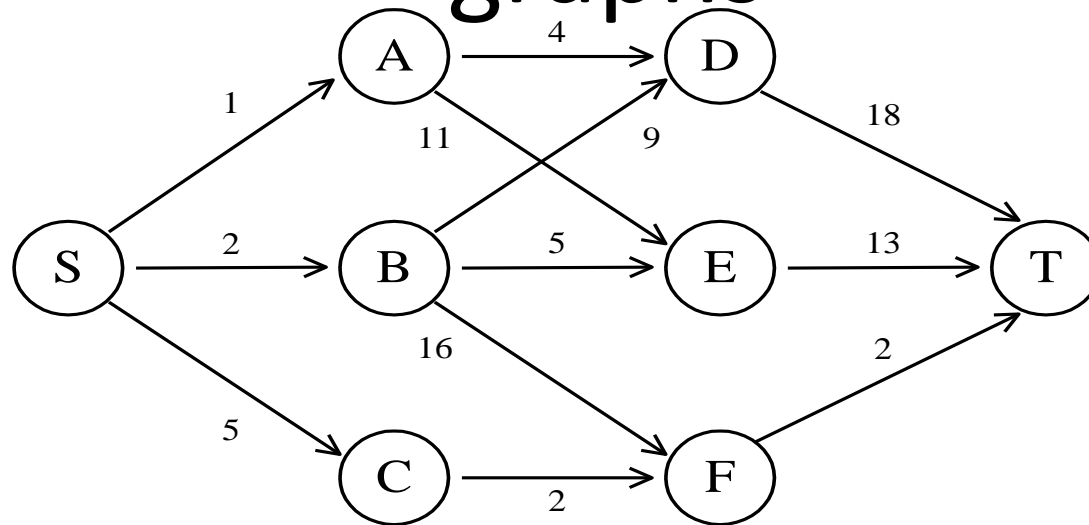
- A multistage graph $G=(V, E)$ is a directed graph, in which
- The vertices are partitioned into $k \geq 2$ disjoint sets $V_i, 1 \leq i < k$
- Sets V_1 and V_k are such that $|V_1| = |V_k| = 1$
- Let s and t be the vertices in V_1 and V_k respectively.
- s is source and t is sink (destination)

Multistage Graph

- Let $c(i, j)$ be the cost of edge (i, j)
- The cost of a path from s to t is the sum of the costs of all the edges on the path
- The multistage graph problem is to find a minimum-cost path from s to t
- Each stage V_i defines a stage in the graph
- Constraint is, every path from s to t starts in stage 1, goes to stage 2 and so on
- And terminates in k stage

The shortest path in multistage graphs

- e.g.



- The greedy method can not be applied to this case:
(S, A, D, T) $1+4+18 = 23$.
- The real shortest path is:
(S, C, F, T) $5+2+2 = 9$.

Multistage Graph

- Dynamic Programming Approach
- Such problems can be solved using 2 approaches
- Forward Approach
 - Backward Reasoning
- Backward Approach
 - Forward Reasoning

Multistage Graph- Example

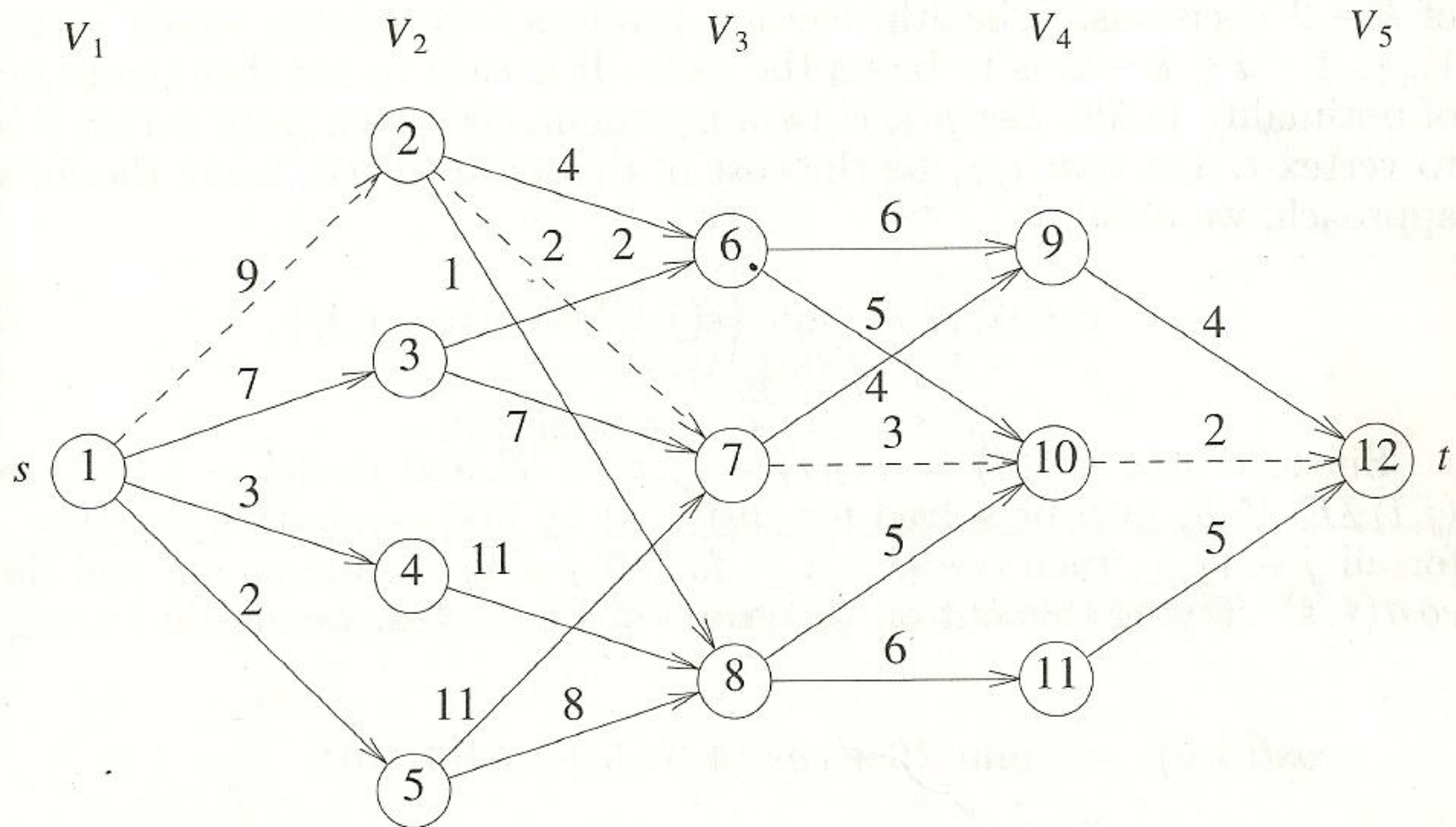


Figure 5.2 Five-stage graph

Multistage Graph- Example

- Every s to t path is the result of a sequence of **k-2** decisions
- i^{th} decision involves determining which vertex in V_{i+1} is to be on the path
- $\text{cost}(i, j) = \min \{c(j, l) + \text{cost}(i+1, l)\}$
- $\text{Cost}(1, 1) = \min \{9 + \text{cost}(2, 2), 7 + \text{cost}(2, 3), 3 + \text{cost}(2, 4), 2 + \text{cost}(2, 5)\}$
- Forward Approach - backward reasoning.
- Calculate Distance from Target as reference

Multistage Graph- Example

- $\text{Cost}(1,1) = \min\{9 + \text{cost}(2,2), 7 + \text{cost}(2,3), 3 + \text{cost}(2,4), 2 + \text{cost}(2,5)\}$
- $\text{Cost}(4,9) = c(9,12) + \text{cost}(5,12)$
 $= 4 + 0 = 4$
- $\text{Cost}(4,10) = c(10,12) + \text{cost}(5,12)$
 $= 2 + 0 = 2$
- $\text{Cost}(4,11) = c(11,12) + \text{cost}(5,12)$
 $= 5 + 0 = 5$

Multistage Graph- Example

- $\text{cost}(3,6) = \min\{6+\text{cost}(4,9), \underline{5+\text{cost}(4,10)}\}$
 $=\min(6+4, 5+2) = \min(10,7)=7$
- $\text{cost}(3,7) = \min\{4+\text{cost}(4,9), \underline{3+\text{cost}(4,10)}\}$
 $=\min(4+4, 3+2)=\min(8,5)=5$
- $\text{cost}(3,8) = \min\{\underline{5+\text{cost}(4,10)}, 6+\text{cost}(4,11)\}$
 $=\min(5+2, 6+5)=\min(7, 11)=7$

Multistage Graph- Example

- $\text{cost}(2,2)$
 $= \min\{4 + \text{cost}(3,6), \underline{2 + \text{cost}(3,7)}, 1 + \text{cost}(3,8)\}$
 $= \min(4+7, 2+5, 1+7) = 7$
- $\text{cost}(2,3)$
 $= \min\{\underline{2 + \text{cost}(3,6)}, 7 + \text{cost}(3,7)\}$
 $= \min(2+7, 7+5) = 9$

Multistage Graph- Example

- $\text{cost}(2,4)$
 $=\min\{11+\text{cost}(3,8)\}$
 $=\min(11+7)=18$
- $\text{cost}(2,5)$
 $=\min\{11+\text{cost}(3,7), \underline{8+\text{cost}(3,8)}\}$
 $=\min(11+5, 8+7)=15$

Multistage Graph- Example

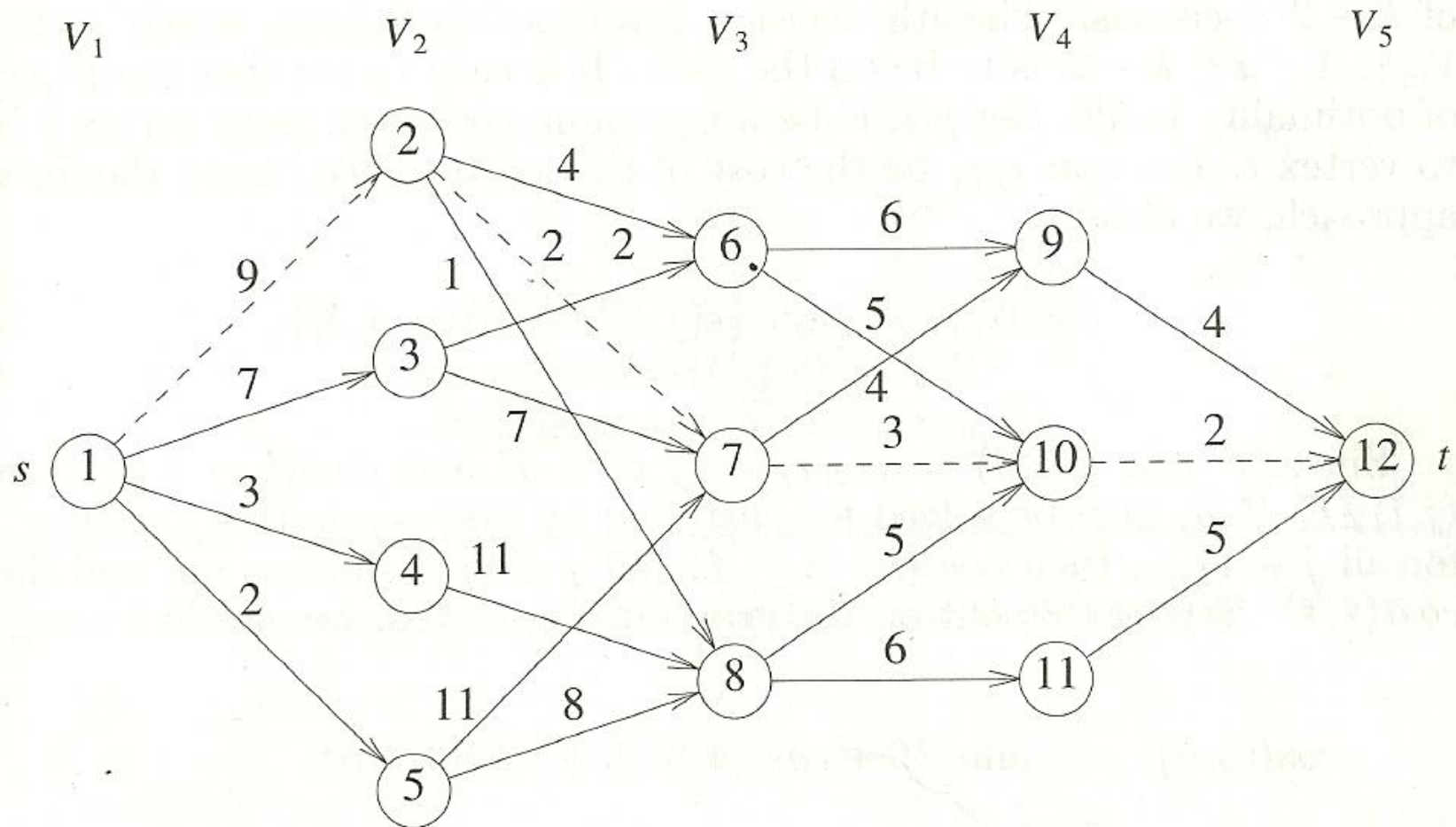


Figure 5.2 Five-stage graph

Multistage Graph- Example

- $\text{Cost}(1,1)$
 $= \min\{\underline{9+\text{cost}(2,2)}, \underline{7+\text{cost}(2,3)}, 3+\text{cost}(2,4),$
 $2+\text{cost}(2,5)\}$
 $= \min(\underline{9+7}, \underline{7+9}, 3+18, 2+15)$
 $= 16$

- A minimum cost s to t path has a cost 16
- This path can be determined easily if we record the decision made at each state (vertex)

Multistage Graph- Example

- Let $d(i, j)$ be the value of I (I is a node in next level) that minimizes $\{c(j, I) + \text{cost}(i+1, I)\}$
- $d(3,6)=10$ $d(3,7)=10$ $d(3,8)=10$
- $d(2,2)=7$ $d(2,3)=6$ $d(2,4)=8$
- $d(2,5)=8$ $d(1,1)=2$ or 3

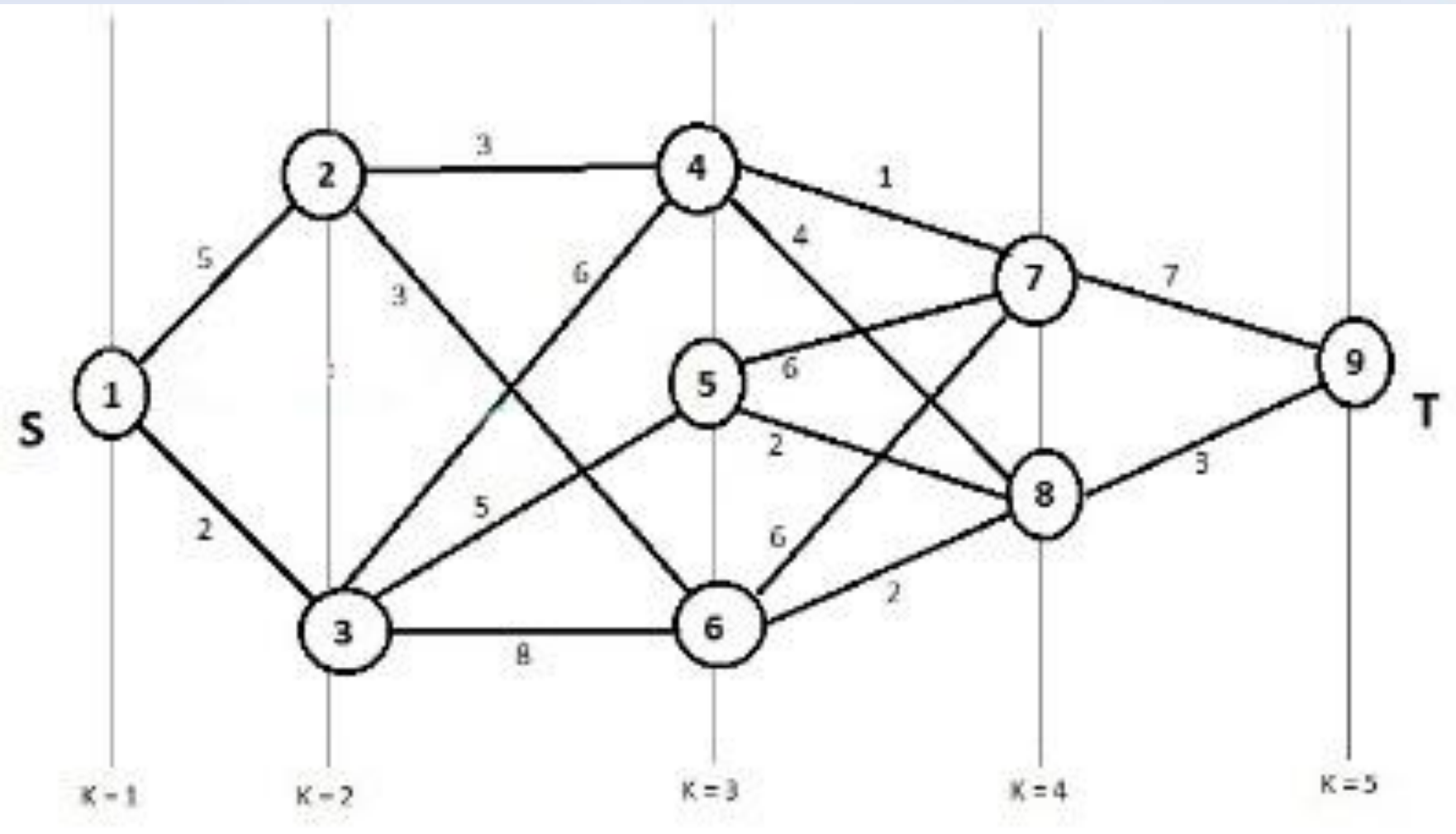
Multistage Graph- Example

- Let the minimum-cost path be
- $s, v_2, v_3, \dots, v_{k-1}, t$
- $V_2 = d(1, 1) = 2$ $v_3 = d(2, 2) = 7$
- $v_4 = d(3, 7) = 10$
- So the path is 1, 2, 7, 10, 12
- Cost = 16

Multistage Graph- Example

- Let the minimum-cost path be
- $s, v_2, v_3, \dots, v_{k-1}, t$
- $V_2 = d(1, 1) = 3$ $v_3 = d(2, 3) = 6$
- $v_4 = d(3, 6) = 10$
- So the path is 1, 3, 6, 10, 12
- Cost = 16

Multistage Graph- Example



Multistage Graph- Backward Approach

- backward Approach - Forward reasoning.
- Calculate Distance from Source as reference
- $\text{bcost}(i, j) = \min(\text{bcost}(i-1, l) + c(l, j))$
- $\text{bcost}(5, 12) = \min\{4 + \text{cost}(4, 9), 2 + \text{cost}(4, 10), 5 + \text{cost}(4, 11)\}$

Multistage Graph- Example

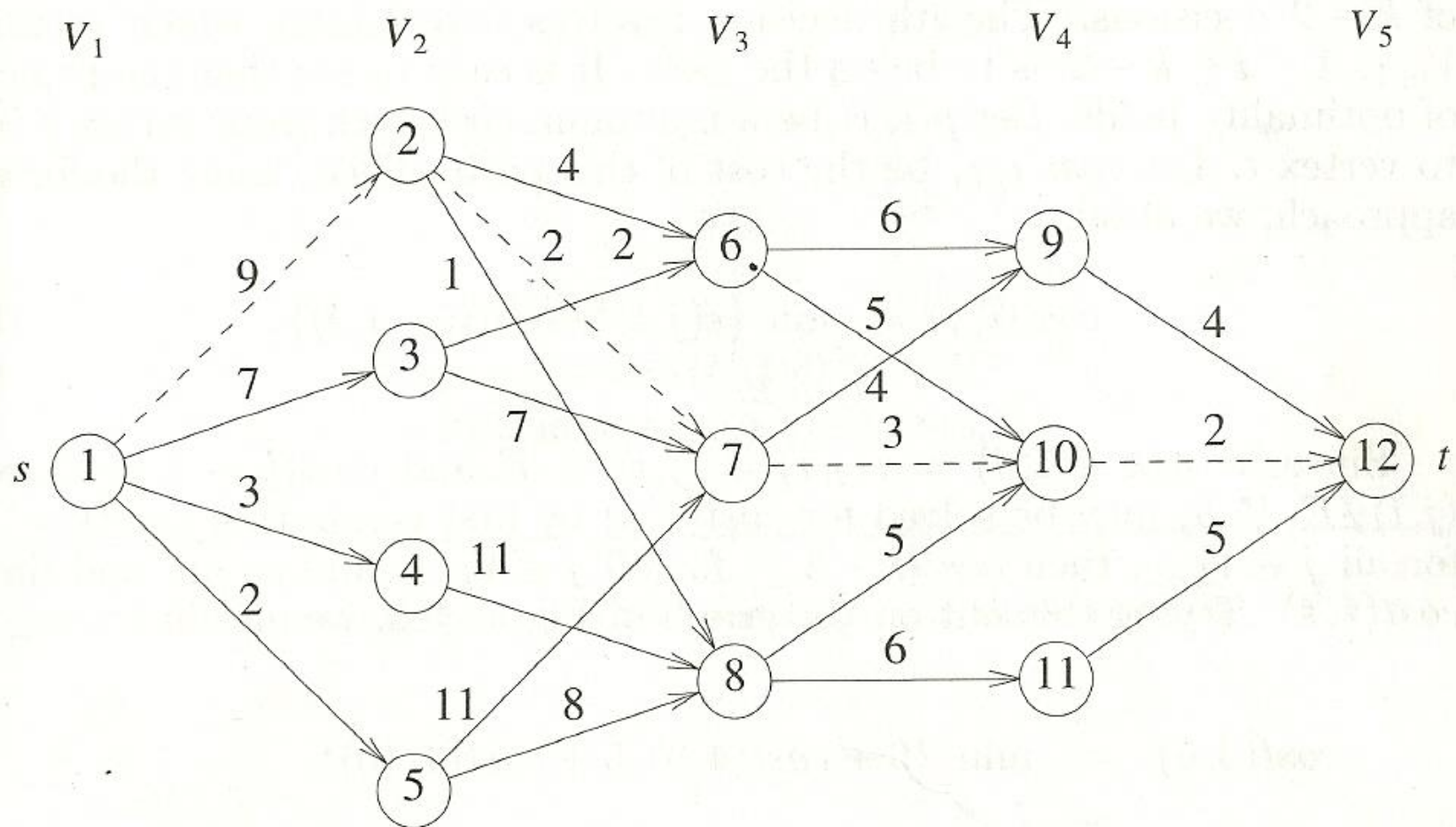


Figure 5.2 Five-stage graph

Multistage Graph- Backward Approach

- $\text{bcost}(i, j) = \min(\text{bcost}(i-1, l) + c(l, j))$
- $\text{bcost}(2, 2) = 9$
- $\text{bcost}(2, 3) = 7$
- $\text{bcost}(2, 4) = 3$
- $\text{bcost}(2, 5) = 2$
- $\text{bcost}(3, 6) = \min\{ \text{bcost}(2, 2) + c(2, 6) \quad \text{bcost}(2, 3) + c(3, 6) \}$
 $= \min\{9 + 4, 7 + 2\} = 9$

Multistage Graph- Backward Approach

- $\text{bcost}(3, 7)=11$
- $\text{bcost}(3, 8)=10$
- $\text{bcost}(4, 9)=15$
- $\text{bcost}(4, 10)=14$
- $\text{bcost}(4, 11)=16$
- $\text{bcost}(5, 12)=16$
-

Multistage Graph

- Example-
- Consider a resource allocation problem
- n units of resource are to be allocated to r projects
- If j , $0 \leq j \leq n$, units of the resource are allocated to project i ,
- Then the resulting net profit is $N(i, j)$

Resource allocation problem

- The problem is to allocate the resource to the r projects
- in such a way as to maximize total net profit
- This problem can be formulated as an $r+1$ stage graph problem
- Stage i , $1 \leq i \leq r$ represents project i
- There are $n+1$ vertices $V(i, j)$, $0 \leq j \leq n$ associated with stage i , $2 \leq i \leq r$

Resource allocation problem

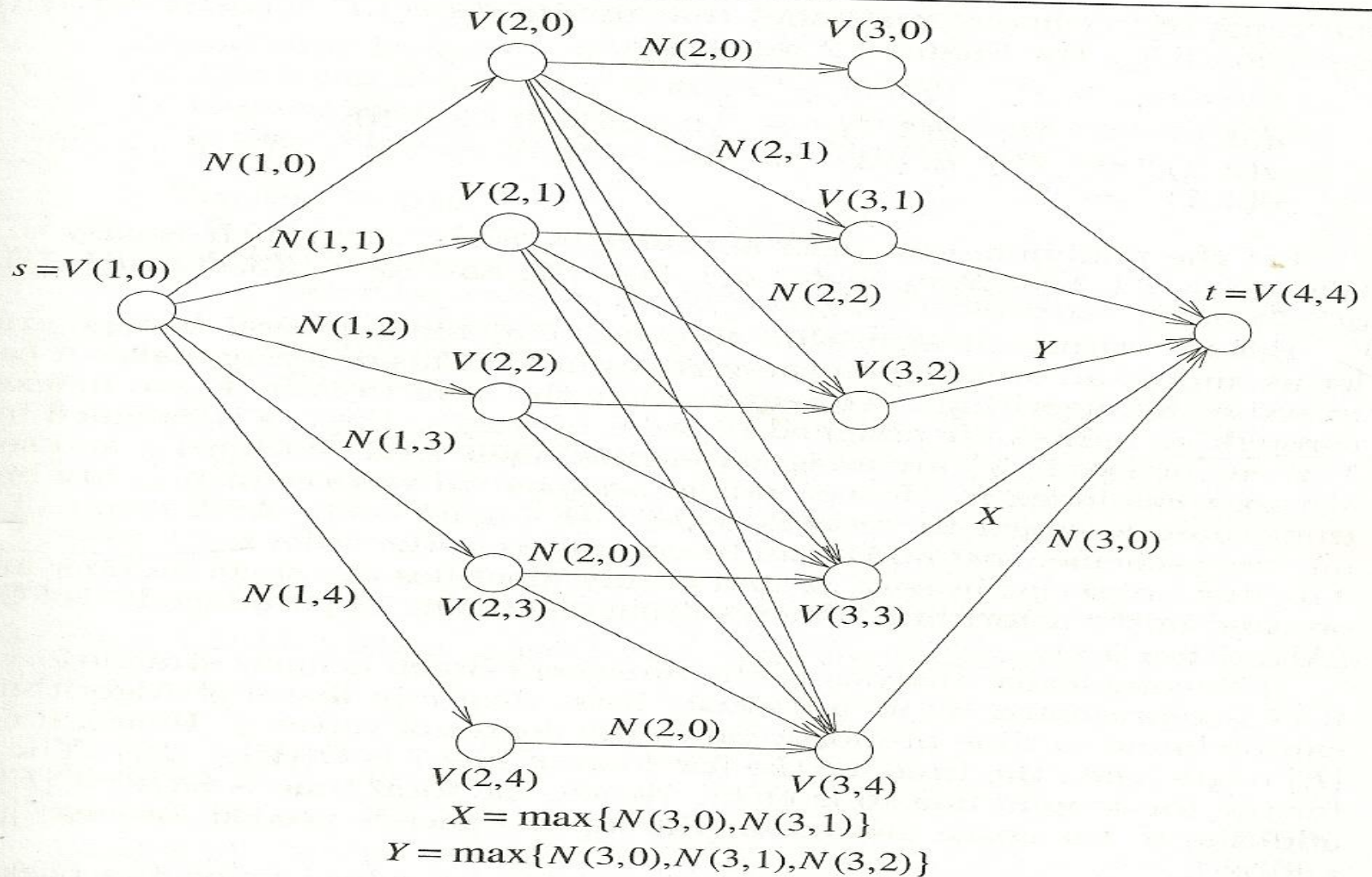


Figure 5.3 Four-stage graph corresponding to a three-project problem

All-Pairs Shortest Paths (APSP)

- Let $G=(V,E)$ be a directed graph with n vertices.
- Let cost be a adjacency matrix for G , such that $\text{cost}(i, i)=0$, $1 \leq i \leq n$
- $\text{cost}(i, j)$ is the length (or cost) of edge (i,j) if $(i, j) \in E(G)$

All-Pairs Shortest Paths (APSP)

- All pair shortest path problem is to determine a matrix A
- such that $A(i, j)$ is the length of shortest path from i to j .
- A can be obtained by solving n Single source shortest path problems
 - Complexity $n \cdot n^2$ i.e. $O(n^3)$

All-Pairs Shortest Paths (APSP)

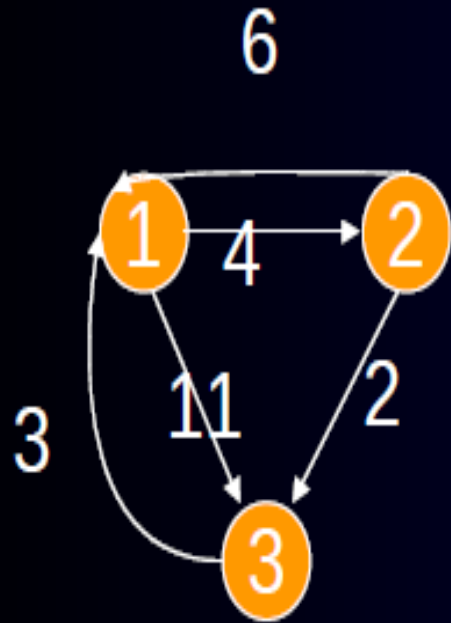
Shortest path from i to j , $i \neq j$

- Path may go through some intermediate vertex
- If k is the index of this intermediate vertex then path (i,k) and (k,j) must be shortest path
- If k is the intermediate vertex with highest index then
 - i to k path is the shortest i to k path going through no vertex with index greater than $k-1$
 - So is the k to j path
- We need to decide highest index intermediate vertex k

All-Pairs Shortest Paths (APSP)

- Let $A^k(i, j)$ be length of shortest path from i to j going through no vertex of index greater than k
- $A(i, j) = \min_{1 \leq k \leq n} \{ \min \{ A^{k-1}(i, k) + A^{k-1}(k, j) \}, \text{cost}(i, j) \}$
- A shortest path from i to j may or may not go through highest index vertex
- $A^k(i, j) = \min \{ A^{k-1}(i, j), A^{k-1}(i, k) + A^{k-1}(k, j) \}$
- $K \geq 1$

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



$$A^0$$

	1	2	3
1	0	4	11
2	6	0	2
3	3	3	∞

$$A^1$$

	1	2	3
1	0	4	11
2	6	0	2
3	3	7	0

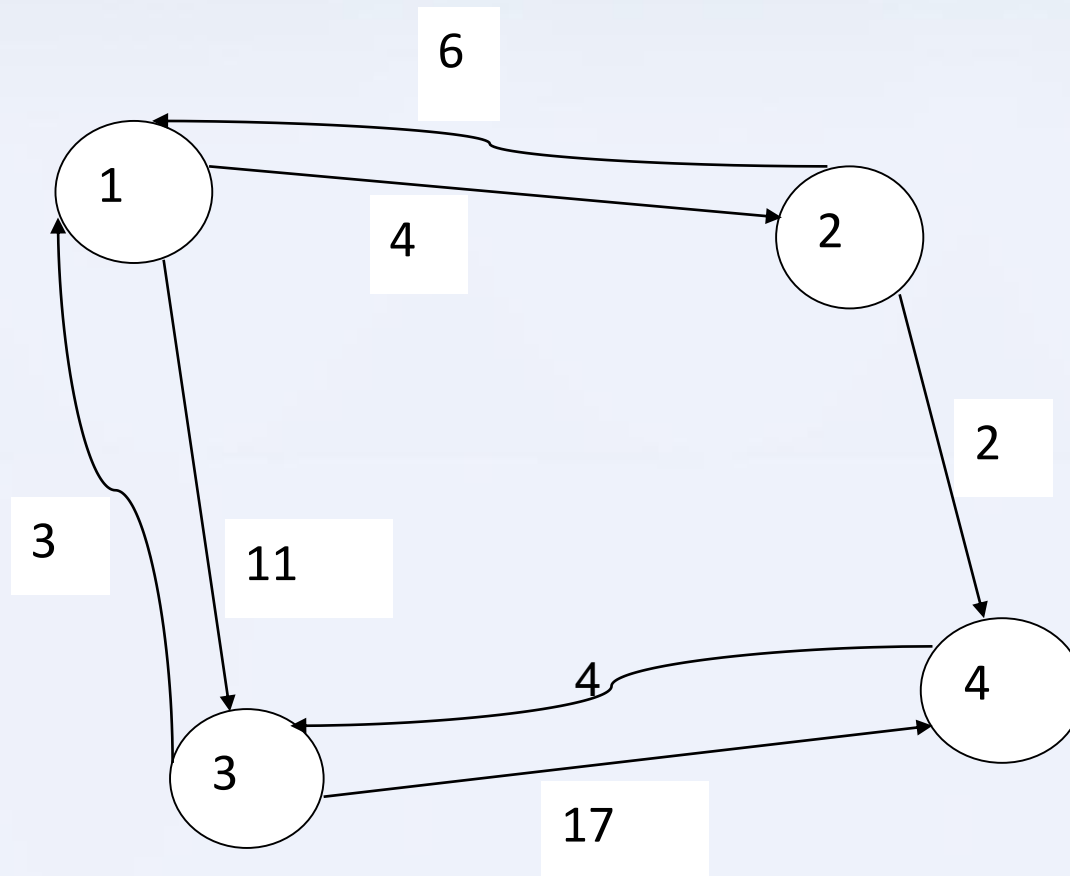
$$A^2$$

	1	2	3
1	0	4	6
2	6	0	2
3	3	7	0

$$A^3$$

	1	2	3
1	0	4	6
2	5	0	2
3	3	7	0

All-Pairs Shortest Paths (APSP)



All-Pairs Shortest Paths (APSP)

- A^0

	1	2	3	4
1	0	4	11	∞
2	6	0	∞	2
3	3	∞	0	17
4	∞	∞	4	0

All-Pairs Shortest Paths (APSP)

- A^1

	1	2	3	4
1	0	4	11	∞
2	6	0	17	2
3	3	7	0	17
4	∞	∞	4	0

All-Pairs Shortest Paths (APSP)

- A^2

	1	2	3	4
1	0	4	11	6
2	6	0	17	2
3	3	7	0	9
4	∞	∞	4	0

All-Pairs Shortest Paths (APSP)

- A^3

	1	2	3	4
1	0	4	11	6
2	6	0	17	2
3	3	7	0	9
4	7	11	4	0

All-Pairs Shortest Paths (APSP)

- A^4

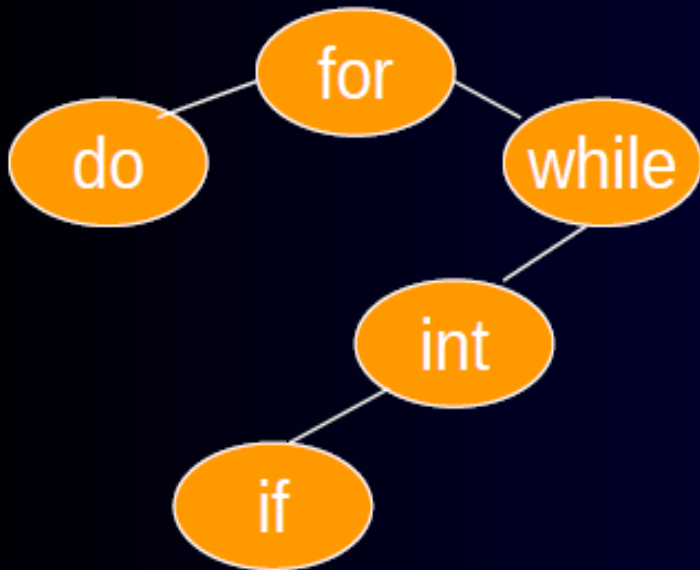
	1	2	3	4
1	0	4	10	6
2	6	0	6	2
3	3	7	0	9
4	7	11	4	0

Optimal Binary Search Tree

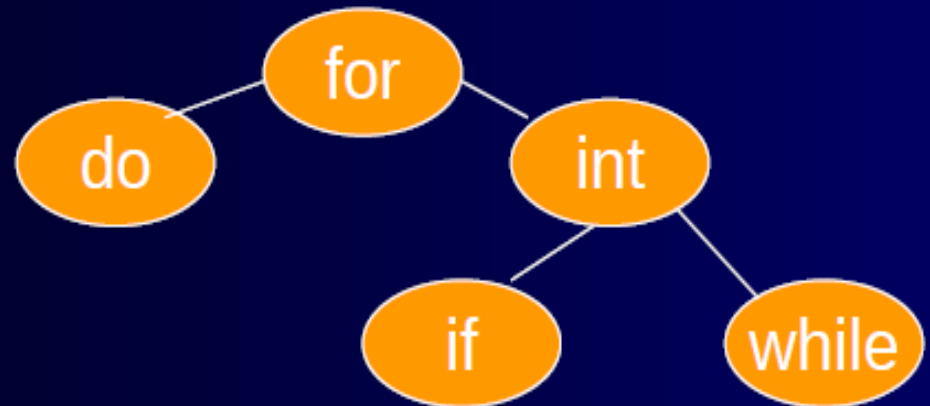
- Given set of identifiers, different binary trees could be formed.
- Average number of comparisons needed for finding the identifiers will be different for different trees.
- In the simplest form we assume
 - Probability of search of each element is equal
 - No unsuccessful searches made

Optimal Binary Search Tree

- Let set of identifiers be {for, do, while, int, if} with $\text{do} < \text{for} < \text{if} < \text{int} < \text{while}$
- Possible trees are



Comparison $= 1 + 2 + 2 + 3 + 4/5$
 $= 12/5$



Comparison $= 1 + 2 + 2 + 3 + 3/5$
 $= 11/5$

Optimal Binary Search Tree

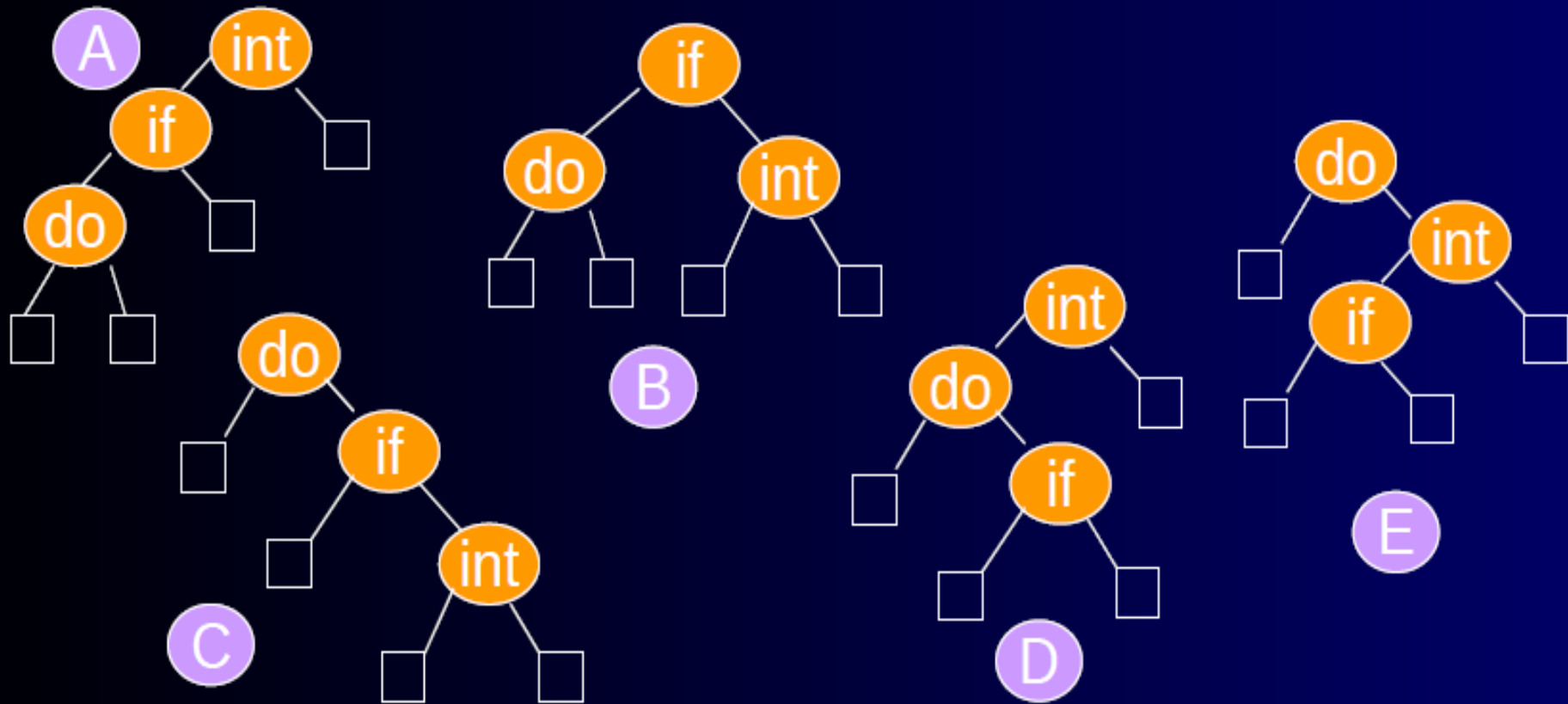
- In general situation different identifiers can be searched with different probabilities
- and unsuccessful searches can also be made
- Let set of identifiers be $\{a_1, a_2, \dots, a_n\}$ with
$$a_1 < a_2 < \dots < a_n$$
- Let $p(i)$ be the probability with which we search a_i .

Optimal Binary Search Tree

- Let $q(i)$ be the probability with which we search x such that $a_i < x < a_{i+1}$, $0 \leq i \leq n$.
- Assume $a_0 = -\infty$ and $a_{n+1} = \infty$.
- Then
- $\sum_{0 \leq i \leq n} q(i)$ is the probability of unsuccessful search
- Clearly
- $\sum_{1 \leq i \leq n} p(i) + \sum_{0 \leq i \leq n} q(i) = 1$

Optimal Binary Search Tree

$$\text{cost} = \sum_{1 \leq i \leq n} p(i) * \text{level}(a_i) + \sum_{0 \leq i \leq n} q(i) * (\text{level}(E_i) - 1)$$



Optimal Binary Search Tree

- $(a_1, a_2, a_3) = (\text{do}, \text{if}, \text{int})$
- If all searches (successful and un-successful) equally probable ($1/7$)
- the cost will be
- $\text{Cost}(\text{tree A}) = 1/7 + 2/7 + 3/7 + 1/7 + 2/7 + 3/7 + 3/7$
- $= 15/7$

$$\text{Cost(B)} = 13/7$$

$$\text{Cost(C)} = 15/7$$

$$\text{Cost(D)} = 15/7$$

$$\text{Cost(E)} = 15/7$$

- Tree B is optimal

Optimal Binary Search Tree

- $(a_1, a_2, a_3) = (\text{do}, \text{if}, \text{int})$
- $p(1)=0.5, p(2)=0.1, p(3)=0.05$ and
- $q(0)=0.15, q(1)=0.1, q(2)=0.05, q(3)=0.05$
- the costs will be
- $\text{cost}(A) = ((0.5 \times 3) + (0.1 \times 2) + (0.05 \times 1)) + ((0.15 \times 3) + (0.1 \times 3) + (0.05 \times 2) + (0.05 \times 1))$
- $= 1.5 + 0.2 + 0.05 + 0.45 + 0.3 + 0.1 + 0.05$
- $= 2.65$

Optimal Binary Search Tree

- $\text{cost}(B) = ((0.5 \times 2) + (0.1 \times 1) + (0.05 \times 2)) + ((0.15 \times 2) + (0.1 \times 2) + (0.05 \times 2) + (0.05 \times 2))$
 $= 1.0 + 0.1 + 0.1 + 0.30 + 0.2 + 0.1 + 0.1$
 $= 1.9$

- $\text{cost}(C) = ((0.5 \times 1) + (0.1 \times 2) + (0.05 \times 3)) + ((0.15 \times 1) + (0.1 \times 2) + (0.05 \times 3) + (0.05 \times 3))$
 $= 0.5 + 0.2 + 0.15 + 0.15 + 0.2 + 0.15 + 0.15$
 $= 1.5$

Optimal Binary Search Tree

- $\text{cost}(D) = ((0.5 \times 2) + (0.1 \times 3) + (0.05 \times 1)) + ((0.15 \times 2) + (0.1 \times 3) + (0.05 \times 3) + (0.05 \times 1))$
 $= 1.0 + 0.3 + 0.05 + 0.30 + 0.3 + 0.15 + 0.05$
 $= 2.15$

- $\text{cost}(E) = ((0.5 \times 1) + (0.1 \times 3) + (0.05 \times 2)) + ((0.15 \times 1) + (0.1 \times 3) + (0.05 \times 3) + (0.05 \times 2))$
 $= 0.5 + 0.3 + 0.10 + 0.15 + 0.3 + 0.15 + 0.10$
 $= 1.6$

Optimal Binary Search Tree

- $p(1)=0.5$, $p(2)=0.1$, $p(3)=0.05$ and
- $q(0)=0.15$, $q(1)=0.1$, $q(2)=0.05$, $q(3)=0.05$
- the costs will be
- $\text{cost}(A)=2.65$
- $\text{cost}(B)=1.9$
- $\text{cost}(C)=1.5$
- $\text{cost}(D)=2.15$
- $\text{cost}(E)=1.6$
- Tree C is optimal

Applying Dynamic Programming Approach

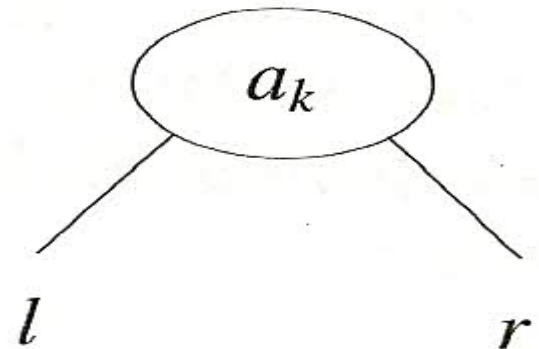
- Construct tree as a result of sequence of decisions
- Make decision as to which of the a_i should be assigned to the root node of the tree
- If we choose a_k as a root, then
- Internal nodes a_1, a_2, \dots, a_{k-1} and external nodes E_1, E_2, \dots, E_{k-1} will lie in left sub-tree l of root
- The remaining nodes will lie in right sub-tree r

Applying Dynamic Programming Approach

$$cost(l) = \sum_{1 \leq i < k} p(i) * level(a_i) + \sum_{0 < i < k} q(i) * (level(E_i) - 1)$$

$$cost(r) = \sum_{k < i \leq n} p(i) * level(a_i) + \sum_{k < i \leq n} q(i) * (level(E_i) - 1)$$

Root of sub-tree is
assumed to be present at
level 1



Applying Dynamic Programming Approach

Using $w(i, j)$ to represent the sum $q(i) + \sum_{l=i+1}^j (q(l) + p(l))$, we obtain the following as the expected cost of the search tree

$$p(k) + \text{cost}(l) + \text{cost}(r) + w(0, k-1) + w(k, n)$$

- If the tree is optimal, then this value must be minimum
- Hence $\text{cost}(l)$ and $\text{cost}(r)$ must be minimum

Applying Dynamic Programming Approach

- If we use $c(i, j)$ to represent the cost of an optimal binary search tree t_{ij} containing $a_{i+1}, a_{i+2}, \dots, a_j$ and E_i, E_{i+1}, \dots, E_j
- Then for the tree to be optimal
- $\text{cost}(l) = c(0, k-1)$ and $\text{cost}(r) = c(k, n)$
- k must be chosen such that

$$p(k) + c(0, k-1) + c(k, n) + w(0, k-1) + w(k, n)$$

is minimum

Applying Dynamic Programming Approach

- Hence for $c(0, n)$ we obtain

$$c(0, n) = \min_{1 \leq k \leq n} \{c(0, k-1) + c(k, n) + p(k) + w(0, k-1) + w(k, n)\}$$

- We can generalize it to obtain for any $c(i, j)$

$$c(i, j) = \min_{i < k \leq j} \{c(i, k-1) + c(k, j) + p(k) + w(i, k-1) + w(k, j)\}$$

$$c(i, j) = \min_{i < k \leq j} \{c(i, k-1) + c(k, j)\} + w(i, j)$$

Applying Dynamic Programming Approach

- This equation can be solved for $c(0, n)$
- By first computing all $c(i, j)$, such that $j-i = 1$
- $c(i, i) = 0$ $w(i, i) = q(i)$ $0 \leq i \leq n$
- Next computing all $c(i, j)$, such that $j-i = 2$ and then computing all $c(i, j)$, such that $j-i = 3$ so on
- During this computation we record the root $r(i, j)$ of each tree t_{ij} ,
- then optimal binary search tree can be constructed from these $r(i, j)$

Example 5.18 Let $n = 4$ and $(a_1, a_2, a_3, a_4) = (\text{do}, \text{if}, \text{int}, \text{while})$. Let $p(1 : 4) = (3, 3, 1, 1)$ and $q(0 : 4) = (2, 3, 1, 1, 1)$. The p 's and q 's have been multiplied by 16 for convenience. Initially, we have $w(i, i) = q(i)$, $c(i, i) = 0$ and $r(i, i) = 0$, $0 \leq i \leq 4$. Using Equation 5.12 and the observation $w(i, j) = p(j) + q(j) + w(i, j - 1)$, we get

$$w(0, 1) = p(1) + q(1) + w(0, 0) = 8$$

$$c(0, 1) = w(0, 1) + \min\{c(0, 0) + c(1, 1)\} = 8$$

$$r(0, 1) = 1$$

$$w(1, 2) = p(2) + q(2) + w(1, 1) = 7$$

$$c(1, 2) = w(1, 2) + \min\{c(1, 1) + c(2, 2)\} = 7$$

$$r(0, 2) = 2$$

$$w(2, 3) = p(3) + q(3) + w(2, 2) = 3$$

$$c(2, 3) = w(2, 3) + \min\{c(2, 2) + c(3, 3)\} = 3$$

$$r(2, 3) = 3$$

$$w(3, 4) = p(4) + q(4) + w(3, 3) = 3$$

$$c(3, 4) = w(3, 4) + \min\{c(3, 3) + c(4, 4)\} = 3$$

$$r(3, 4) = 4$$

	0	1	2	3	4
0	$w_{00} = 2$ $c_{00} = 0$ $r_{00} = 0$	$w_{11} = 3$ $c_{11} = 0$ $r_{11} = 0$	$w_{22} = 1$ $c_{22} = 0$ $r_{22} = 0$	$w_{33} = 1$ $c_{33} = 0$ $r_{33} = 0$	$w_{44} = 1$ $c_{44} = 0$ $r_{44} = 0$
1	$w_{01} = 8$ $c_{01} = 8$ $r_{01} = 1$	$w_{12} = 7$ $c_{12} = 7$ $r_{12} = 2$	$w_{23} = 3$ $c_{23} = 3$ $r_{23} = 3$	$w_{34} = 3$ $c_{34} = 3$ $r_{34} = 4$	
2	$w_{02} = 12$ $c_{02} = 19$ $r_{02} = 1$	$w_{13} = 9$ $c_{13} = 12$ $r_{13} = 2$	$w_{24} = 5$ $c_{24} = 8$ $r_{24} = 3$		
3	$w_{03} = 14$ $c_{03} = 25$ $r_{03} = 2$	$w_{14} = 11$ $c_{14} = 19$ $r_{14} = 2$			
4	$w_{04} = 16$ $c_{04} = 32$ $r_{04} = 2$				

Applying Dynamic Programming Approach

- $w(i, j) = p(j) + q(j) + w(i, j-1)$

$$c(i, j) = \min_{i < k \leq j} \{c(i, k-1) + c(k, j)\} + w(i, j)$$

- $w(0, 2) = p(2) + q(2) + w(0, 1)$

$$= 3 + 1 + 8$$

$$= 12$$

$$c(0, 2) = \min\{c(0, 0) + c(1, 2), c(0, 1) + c(2, 2)\} + w(0, 2)$$

$$= \min\{0 + 3, 7 + 0\} + 12$$

$$= 3 + 12 = 15$$

$$r(0, 2) = 1;$$

Applying Dynamic Programming Approach

- $w(i, j) = p(j) + q(j) + w(i, j-1)$

$$c(i, j) = \min_{i < k \leq j} \{c(i, k-1) + c(k, j)\} + w(i, j)$$

- $w(1, 3) = p(3) + q(3) + w(1, 2)$

$$= 1 + 1 + 7$$

$$= 9$$

$$c(1, 3) = \min\{c(1, 1) + c(2, 3), c(1, 2) + c(3, 3)\} + w(1, 3)$$

$$= \min\{0 + 3, 7 + 0\} + 9$$

$$= 3 + 9 = 12$$

$$r(1, 3) = 2;$$

Applying Dynamic Programming Approach

- $w(i, j) = p(j) + q(j) + w(i, j-1)$

$$c(i, j) = \min_{i < k \leq j} \{c(i, k-1) + c(k, j)\} + w(i, j)$$

- $w(2, 4) = p(4) + q(4) + w(2, 3)$

$$= 1 + 1 + 3$$

$$= 5$$

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

$$= \min\{0 + 3, 3 + 0\} + 5$$

$$= 3 + 5 = 8$$

$$r(2, 4) = 3;$$

Applying Dynamic Programming Approach

$$w(0, 3) = p(3) + q(3) + w(0, 2)$$

$$= 1 + 1 + 12$$

$$= 14$$

$$c(0, 3) = \min\{c(0, 0) + c(1, 3), c(0, 1) + c(2, 3), \\ c(0, 2) + c(3, 3)\} + w(0, 3)$$

$$= \min\{0 + 12, 8 + 3, 19 + 0\} + 14$$

$$= 11 + 14 = 25$$

$$r(0, 3) = 2;$$

Applying Dynamic Programming Approach

$$w(1, 4) = p(4) + q(4) + w(1, 3)$$

$$= 1 + 1 + 9$$

$$= 11$$

$$c(1, 4) = \min\{c(1, 1) + c(2, 4), c(1, 2) + c(3, 4), \\ c(1, 3) + c(4, 4)\} + w(1, 4)$$

$$= \min\{0 + 8, 7 + 3, 12 + 0\} + 11$$

$$= 8 + 11 = 19$$

$$r(1, 4) = 2;$$

Applying Dynamic Programming Approach

$$w(0, 4) = p(4) + q(4) + w(0, 3)$$

$$= 1 + 1 + 14$$

$$= 16$$

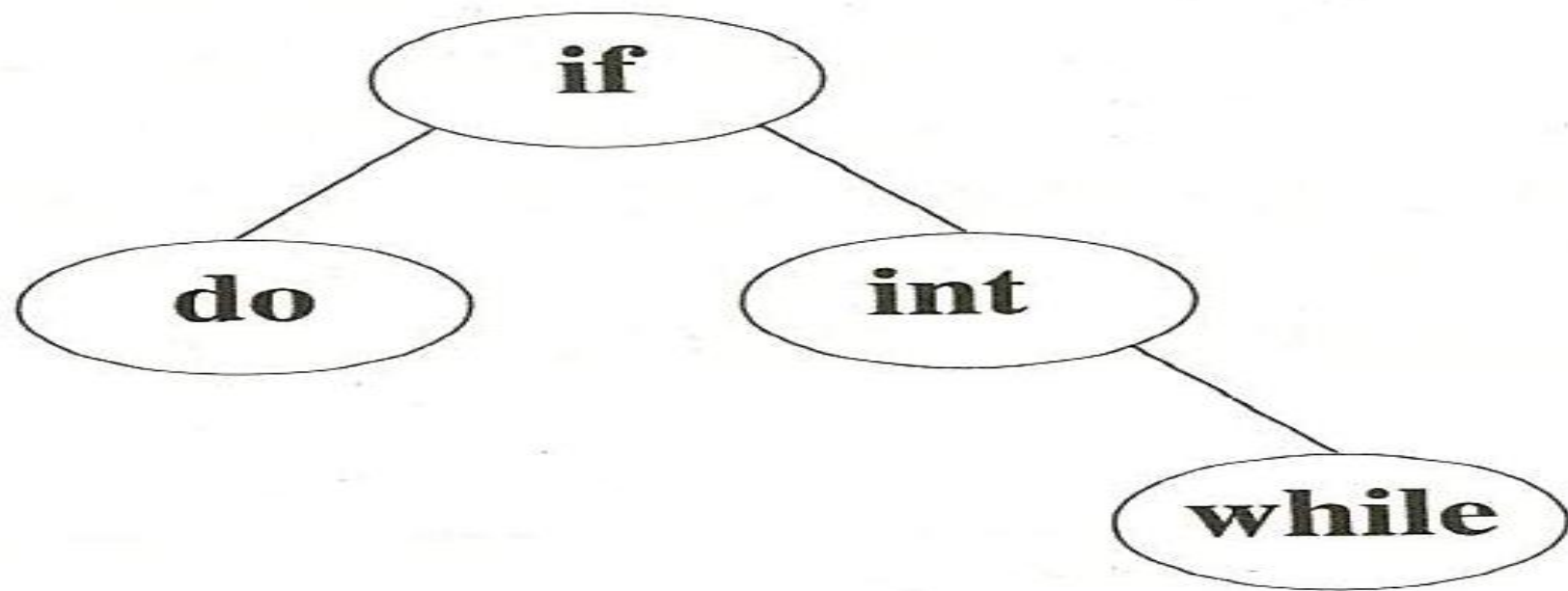
$$r(0, 4) = \min\{c(0, 0) + c(1, 4), c(0, 1) + c(2, 4), \\ c(0, 2) + c(3, 4), c(0, 3) + c(4, 4)\} + w(0, 4)$$

$$= \min\{0 + 19, 8 + 8, 19 + 3, 25 + 0\} + 16$$

$$= 16 + 16 = 32$$

$$r(0, 4) = 2;$$

	0	1	2	3	4
0	$w_{00} = 2$ $c_{00} = 0$ $r_{00} = 0$	$w_{11} = 3$ $c_{11} = 0$ $r_{11} = 0$	$w_{22} = 1$ $c_{22} = 0$ $r_{22} = 0$	$w_{33} = 1$ $c_{33} = 0$ $r_{33} = 0$	$w_{44} = 1$ $c_{44} = 0$ $r_{44} = 0$
1	$w_{01} = 8$ $c_{01} = 8$ $r_{01} = 1$	$w_{12} = 7$ $c_{12} = 7$ $r_{12} = 2$	$w_{23} = 3$ $c_{23} = 3$ $r_{23} = 3$	$w_{34} = 3$ $c_{34} = 3$ $r_{34} = 4$	
2	$w_{02} = 12$ $c_{02} = 19$ $r_{02} = 1$	$w_{13} = 9$ $c_{13} = 12$ $r_{13} = 2$	$w_{24} = 5$ $c_{24} = 8$ $r_{24} = 3$		
3	$w_{03} = 14$ $c_{03} = 25$ $r_{03} = 2$	$w_{14} = 11$ $c_{14} = 19$ $r_{14} = 2$			
4	$w_{04} = 16$ $c_{04} = 32$ $r_{04} = 2$				



compute $w(i, j)$, $r(i, j)$, and $c(i, j)$, $0 \leq i < j \leq 4$, for the identifier set $(a_1, a_2, a_3, a_4) = (\text{cout}, \text{float}, \text{if}, \text{while})$ with $p(1) = 1/20$, $p(2) = 1/5$, $p(3) = 1/10$, $p(4) = 1/20$, $q(0) = 1/5$, $q(1) = 1/10$, $q(2) = 1/5$, $q(3) = 1/20$, and $q(4) = 1/20$. Using the $r(i, j)$'s, construct the optimal binary search tree.

0 / 1 Knapsack

Statement: Same as knapsack except that no fraction of item is allowed (0 or 1 is allowed).

- Solution obtained by sequence of decision x_1, x_2, \dots, x_n ($x = 0/1$).
- Let $f_i(y)$ be the value of optimal solution to $\text{KNAP}(1, i, y)$
 - $f_i(y) = \max \{f_{i-1}(y), f_{i-1}(y - w_i) + p_i\}$.
- S^i be the pair (P, W) where $P = f_i(y_j)$ and $W = y_j$
- S^{i+1} can be computed from S^i by computing $S_1^i = \{(P, W) | (P - p_i, W - w_i) \in S^i\}$
- S^{i+1} can be computed by merging the pairs S^i and S_1^i together.

0 / 1 Knapsack

- If S^{i+1} contains two pairs (P_j, W_j) and (P_k, W_k) with the values $P_j \leq P_k$ and $W_j \geq W_k$
- Then pair (P_j, W_j) can be discarded
- It is called Dominance Rule/ Purging Rule.

Example 5.21 Consider the knapsack instance $n = 3$, $(w_1, w_2, w_3) = (2, 3, 4)$, $(p_1, p_2, p_3) = (1, 2, 5)$, and $m = 6$. For these data we have

$$S^0 = \{(0, 0)\}; S_1^0 = \{(1, 2)\}$$

$$S^1 = \{(0, 0), (1, 2)\}; S_1^1 = \{(2, 3), (3, 5)\}$$

$$S^2 = \{(0, 0), (1, 2), (2, 3), (3, 5)\}; S_1^2 = \{(5, 4), (6, 6), (7, 7), (8, 9)\}$$

$$S^3 = \{(0, 0), (1, 2), (2, 3), (5, 4), (6, 6), (7, 7), (8, 9)\}$$

Note that the pair $(3, 5)$ has been eliminated from S^3 as a result of the purging rule stated above.

0 / 1 Knapsack

Example 5.22 With $m = 6$, the value of $f_3(6)$ is given by the tuple $(6, 6)$ in S^3 (Example 5.21). The tuple $(6, 6) \notin S^2$, and so we must set $x_3 = 1$. The pair $(6, 6)$ came from the pair $(6 - p_3, 6 - w_3) = (1, 2)$. Hence $(1, 2) \in S^2$. Since $(1, 2) \in S^1$, we can set $x_2 = 0$. Since $(1, 2) \notin S^0$, we obtain $x_1 = 1$. Hence an optimal solution is $(x_1, x_2, x_3) = (1, 0, 1)$. \square

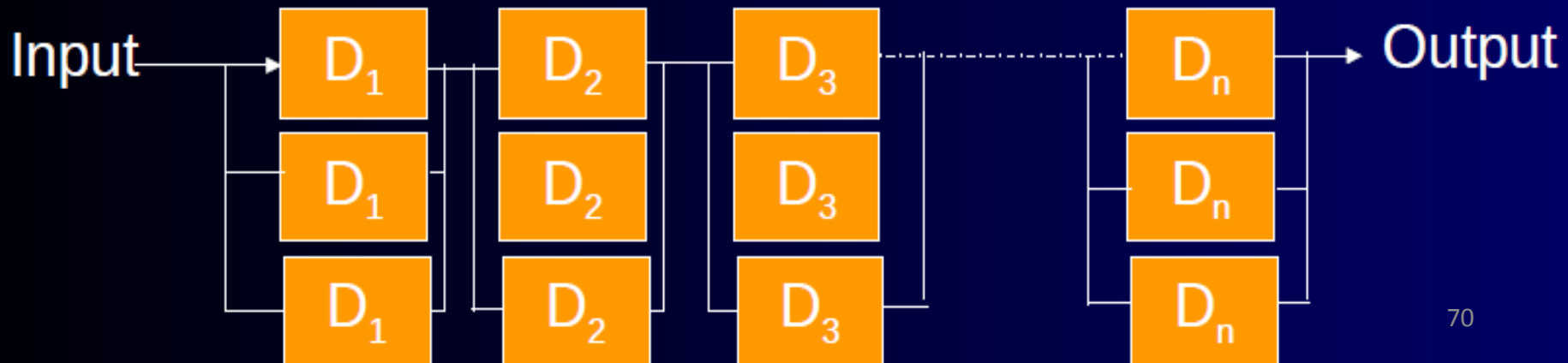
Generate the sets S^i , $0 \leq i \leq 4$, when
(w_1, w_2, w_3, w_4) = (10, 15, 6, 9) and
(p_1, p_2, p_3, p_4) = (2, 5, 8, 1)
 $m=25$

Reliability Design

Problem statement: To design a system that is composed of several devices connected in series.



If each device is associated with the reliability ($r_1, r_2, r_3, \dots, r_n$), the reliability of entire system will be $\prod r_n$. This can be improved by connecting more number of devices in parallel.



Reliability Design: Statement

- If m_i no of devices are connected in parallel then the reliability will be $1-(1-r_i)^{m_i}$
- Let reliability of i^{th} stage is given as $\phi_i(m_i)$
- Reliability of the system is
- Let c_i be cost of each unit and c total cost.
- We want to maximize the above expression subject to

$$\sum_{1 \leq i \leq n} c_i(m_i) \leq c$$

Reliability Design: Solution

$$f_i(x) = \max_{1 \leq m \leq u_i} \{ \phi_i(m_i) f_{i-1}(x - c_i m_i) \}$$

- If $f_0(x)=1$ for all x , $0 \leq x \leq c$.
- Let S^i consists of tuples from (f, x) , with $f=f_i(x)$
- There is at most one tuple for each different x that result from sequence of decision on m_1, m_2, \dots, m_n
- Dominance rule
 (f_1, x_1) dominates (f_2, x_2) iff $f_1 \geq f_2$ and $x_1 \leq x_2$
 Dominated tuples can be discarded

Reliability Design: Example

- The devices be D_1, D_2, D_3 with costs as $\{30, 15, 20\}$ with reliability $\{0.9, 0.8, 0.5\}$ and total cost 105.
- Compute m_1, m_2, m_3 and reliability ???

$$c_1 = 30, c_2 = 15, c_3 = 20, c = 105, r_1 = 0.9, r_2 = 0.8, r_3 = 0.5$$

- $u_1 = 2, u_2 = 3, u_3 = 3$
- Let S^i represents set of all undominated tuples (f, x) that may result from various decision sequences m_1, m_2, \dots, m_n
- S^i is computed from S^{i-1}
- S_j^i represents all tuple obtainable from S^{i-1} by choosing $m_i = j$

Reliability Design: Solution

- $S^0 = \{(1,0)\}$
- $c_1 = 30, c_2 = 15, c_3 = 20, c = 105$
 - $r_1 = 0.9, r_2 = 0.8, r_3 = 0.5$
- $S^1_1 = \{(.9, 30)\}, S^1_2 = \{(.99, 60)\}$
- $S^1 = \{(.9, 30), (.99, 60)\}$
- $u_1 = 2, u_2 = 3, u_3 = 3$
- $S^2_1 = \{(.72, 45), (.792, 75)\}, S^2_2 = \{(.864, 60), (.954, 90)\}, S^2_3 = \{(.8928, 75)\},$
 - Tuple $\{(.954, 90)\}$ which comes from $(.99, 60)$ has been eliminated from S^2_2 as it leaves only Rs 10, not sufficient to have $m_3 = 1$.
- So $S^2 = \{(.72, 45), (.864, 60), (.8928, 75)\}$
 - Tuple $(.792, 75)$ is dominated by $(.864, 60)$.
- $S^3_1 = \{(.36, 65), (.423, 80), (.4464, 95)\}, S^3_2 = \{(.54, 85), (.648, 100)\}, S^3_3 = \{(.63, 105)\}$
- $S^3 = \{(.36, 65), (.423, 80), (.54, 85), (.648, 100)\}$
- Best Design with reliability .648 and cost 100
- Tracking back through S^i we get $m_1 = 1, m_2 = 2, m_3 = 2$

Reliability Design

- $s^0 = \{(1, 0)\}$ – Initial Set Before adding any device
- Add 1 device of type D1
- $s^1_1 = \{(0.9, 30)\}$
- Add 2 devices of type D1
- Reliability of 2 devices of type D1 connected in parallel
- $1 - (1 - r)^m = 1 - (1 - 0.9)^2 = 1 - 0.1^2 = 0.99$
- $s^1_2 = \{(0.99, 60)\}$
- $s^1 = \{(0.9, 30), (0.99, 60)\}$

Reliability Design

- Add 1 device of type D2
- $s^2_1 = \{(0.72, 45), (0.792, 75)\}$
- Add 2 devices of type D2
- Reliability of 2 devices of type D2 connected in parallel
- $1 - (1 - r)^m = 1 - (1 - 0.8)^2 = 1 - 0.2^2 = 0.96$
- $s^2_2 = \{(0.864, 60), (0.9504, 90)\}$

Reliability Design

- Add 3 devices of type D2
- Reliability of 3 devices of type D2 connected in parallel
- $1 - (1 - r)^m = 1 - (1 - 0.8)^3 = 1 - 0.2^3 = 0.992$
- $s^2_3 = \{ (0.8928, 75) \}$
- $s^2 = \{ (0.72, 45), (0.792, 75), (0.864, 60), (0.9504, 90), (0.8928, 75) \}$
- $s^2 = \{ (0.72, 45), (0.864, 60), (0.8928, 75) \}$

Reliability Design

- Add 1 device of type D3
- $s^3_1 = \{(0.36, 65), (0.432, 80), (0.4464, 95)\}$
- Add 2 devices of type D3
- Reliability of 2 devices of type D3 connected in parallel
- $1 - (1 - r)^m = 1 - (1 - 0.5)^2 = 1 - 0.5^2 = 0.75$
- $s^3_2 = \{(0.54, 85), (0.648, 100)\}$

Reliability Design

- Add 3 devices of type D3
- Reliability of 3 devices of type D3 connected in parallel
- $1-(1-r)^m = 1-(1-0.5)^3 = 1-0.5^3 = 0.875$
- $s^3_3 = \{ (0.63, 105) \}$
- $s^3 = \{ (0.36, 65), (0.432, 80), (0.54, 85), (0.648, 100) \} (0.63, 105) \}$
- Optimal - (0.648, 100)
- 1,2,2

Reliability Design- Example 2

- $C1=40$ $c2=25$ $c3=35$
- $r1=0.75$ $r2=0.85$ $r3=0.6$
- $u1=3$ $u2=3$ $u3=2$
- $c=175$

- Till now we have seen how dynamic programming problem could be applicable to **subset selection problem**
- Now will see some permutation problem with $n!$ complexity

Traveling Salesperson Problem

- Statement: Let $G(V,E)$ be a directed graph with edge cost c_{ij} .
- $c_{ij} > 0$, and $c_{ij} = \infty$ if $\langle i, j \rangle \notin E$.
- A tour of G is directed simple cycle that includes every vertex in V .
- Cost of tour is sum of cost of edges on the tour

Traveling Salesperson Problem

- Traveling salesman problem is to find tour of minimum cost.

Traveling Salesman Problem: Applications

- Postal van picking mails from postal mail boxes.

Traveling Salesman Problem: Solution

- Let the tour starts at vertex 1 and the next vertex visited is k .
- Tour will consists of edge $\langle 1, k \rangle$ for some $k \in V - \{1\}$ and path from vertex k to 1.
- This path from k to 1 goes through each vertex in $V - \{1, k\}$ exactly once.
- If the tour is optimal, the path from k to 1 must be shortest k to 1 path, going through all vertex in $V - \{1, k\}$.
- Let $g(i, S)$ be the length of shortest path starting at vertex i , going through all vertex in S and terminating at vertex 1.

Traveling Salesman Problem: Solution

- The function $g(1, V - \{1, k\})$ is the length an optimal salesperson tour.

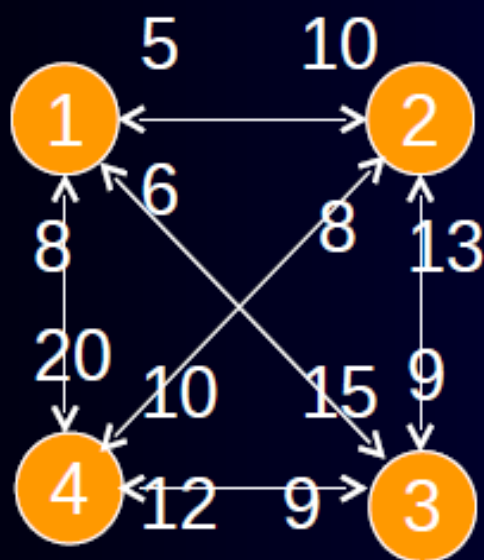
- Generalizing
$$g(1, V - \{1\}) = \min_{2 \leq k \leq n} \{c_{1k} + g(k, V - \{1, k\})\}$$

$$g(i, V - \{1\}) = \min_{j \in S} \{c_{ij} + g(i, S - \{j\})\}$$

- $g(i, \emptyset) = c_{i1},$

Traveling Salesman Problem: Example

$$g(i, V - \{1\}) = \min_{j \in S} \{c_{ij} + g(i, S - \{j\})\}$$



- $g(2, \emptyset) = c_{21} = 5$
- $g(2, \{3\}) = c_{23} + g(3, \emptyset) = 15$
 $g(2, \{4\}) = ?$ $g(3, \{4\}) = ?$
- $g(2, \{3, 4\}) = \min \{c_{23} + g(3, \{4\}), c_{24} + g(4, \{3\})\} = 25$
- $g(3, \{2, 4\}) = ?$
- $g(1, \{2, 3, 4\}) = 35$

0	10	15	20
5	0	9	10
6	13	0	12
8	8	9	0

Traveling Salesman Problem

- $g(2, \Phi) = C_{21} = 5$
- $g(3, \Phi) = C_{31} = 6$
- $g(4, \Phi) = C_{41} = 8$
- $g(2, \{3\}) = C_{23} + g(3, \Phi) = 9 + 6 = 15$
- $g(2, \{4\}) = C_{24} + g(4, \Phi) = 10 + 8 = 18$
- $g(3, \{2\}) = C_{32} + g(2, \Phi) = 13 + 5 = 18$
- $g(3, \{4\}) = C_{34} + g(4, \Phi) = 12 + 8 = 20$
- $g(4, \{2\}) = C_{42} + g(2, \Phi) = 8 + 5 = 13$
- $g(4, \{3\}) = C_{43} + g(3, \Phi) = 9 + 6 = 15$

Traveling Salesman Problem

- $g(2, \{3, 4\}) =$
- $= \min \{C_{23} + g(3, \{4\}), \underline{C_{24} + g(4, \{3\})}\}$
- $= \min \{9+20, 10+15\} = \min\{29, 25\} = 25$
- $g(3, \{2, 4\}) =$
- $= \min \{C_{32} + g(2, \{4\}), \underline{C_{34} + g(4, \{2\})}\}$
- $= \min \{13+18, 12+13\} = \min\{31, 25\} = 25$
- $g(4, \{2, 3\}) =$
- $= \min \{\underline{C_{42} + g(2, \{3\})}, C_{43} + g(3, \{2\})\}$
- $= \min \{8+15, 9+18\} = \min\{23, 27\} = 23$

Traveling Salesman Problem

- $g(1, \{2, 3, 4\}) =$
- $= \min \{ \underline{C_{12} + g(2, \{3, 4\})},$
 $C_{13} + g(3, \{2, 4\}),$
 $C_{14} + g(4, \{2, 3\}) \}$
- $= \min \{ 10+25, 15+25, 20+23 \}$
- $= \min \{ 35, 40, 43 \}$
- $= 35$

$v1 = 1 \quad v2 = 2 \quad v3 = 4 \quad v4 = 3 \quad v5 = 1$

• 1, 2, 4, 3, 1

Flow Shop Scheduling

Problem Statement

- n jobs requiring m tasks
 $T_{1i}, T_{2i}, T_{3i}, \dots, T_{mi}, 1 \leq i \leq n$
- Task T_{ji} is to be performed on Processor $P_j, 1 \leq i \leq m$
- Time required to complete T_{ji} is t_{ji}
- Schedule for n jobs is an assignment of task to time interval on the processor.

Problem statement: Conditions

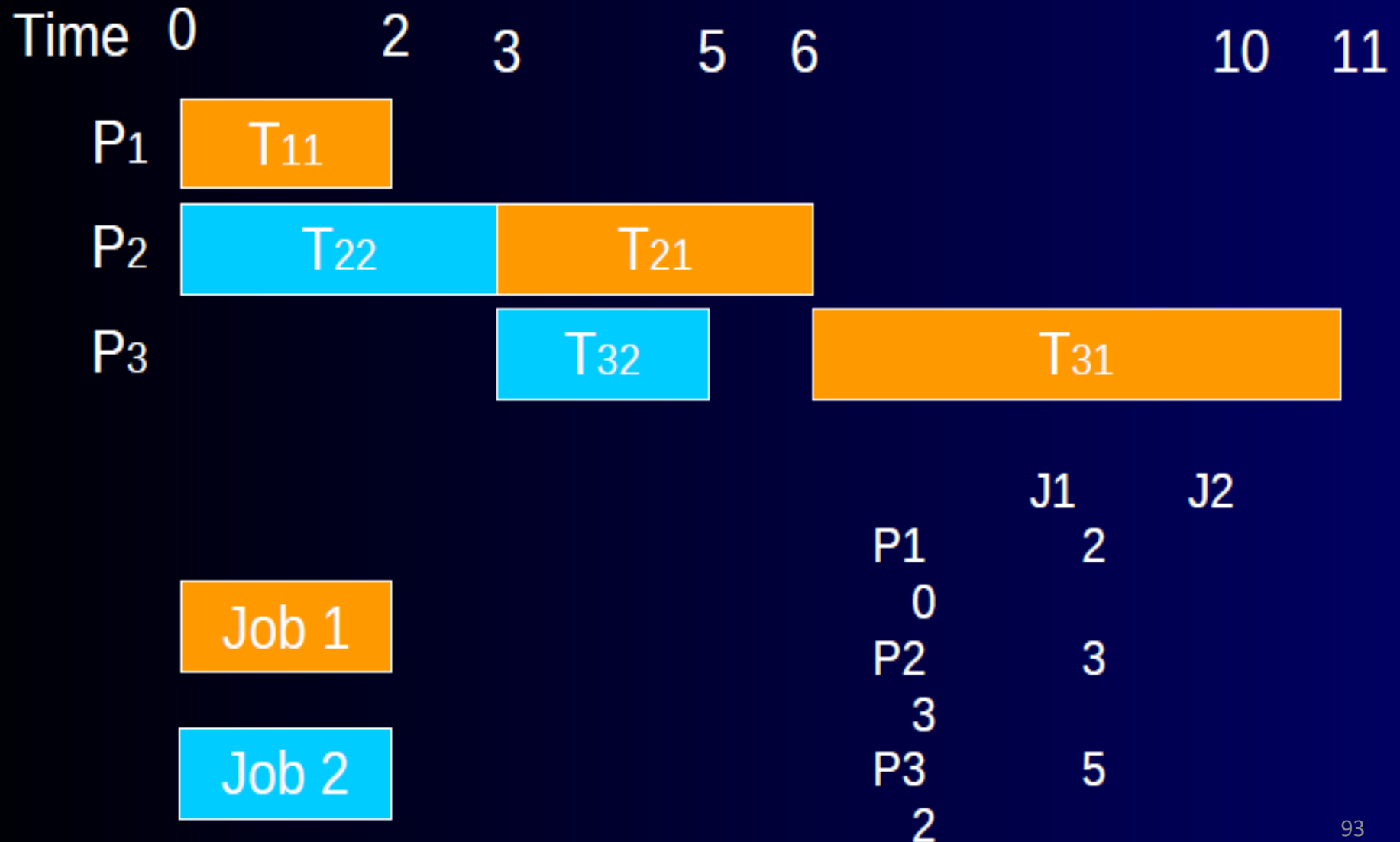
- Task T_{ji} must be assigned to the Processor P_j
- No processor may have more than one task assigned to it in any time interval.
- For any job i the Processing of task $T_{ji}, j \geq 1$, cannot be started until task $T_{j-1,i}$ is completed.

Example

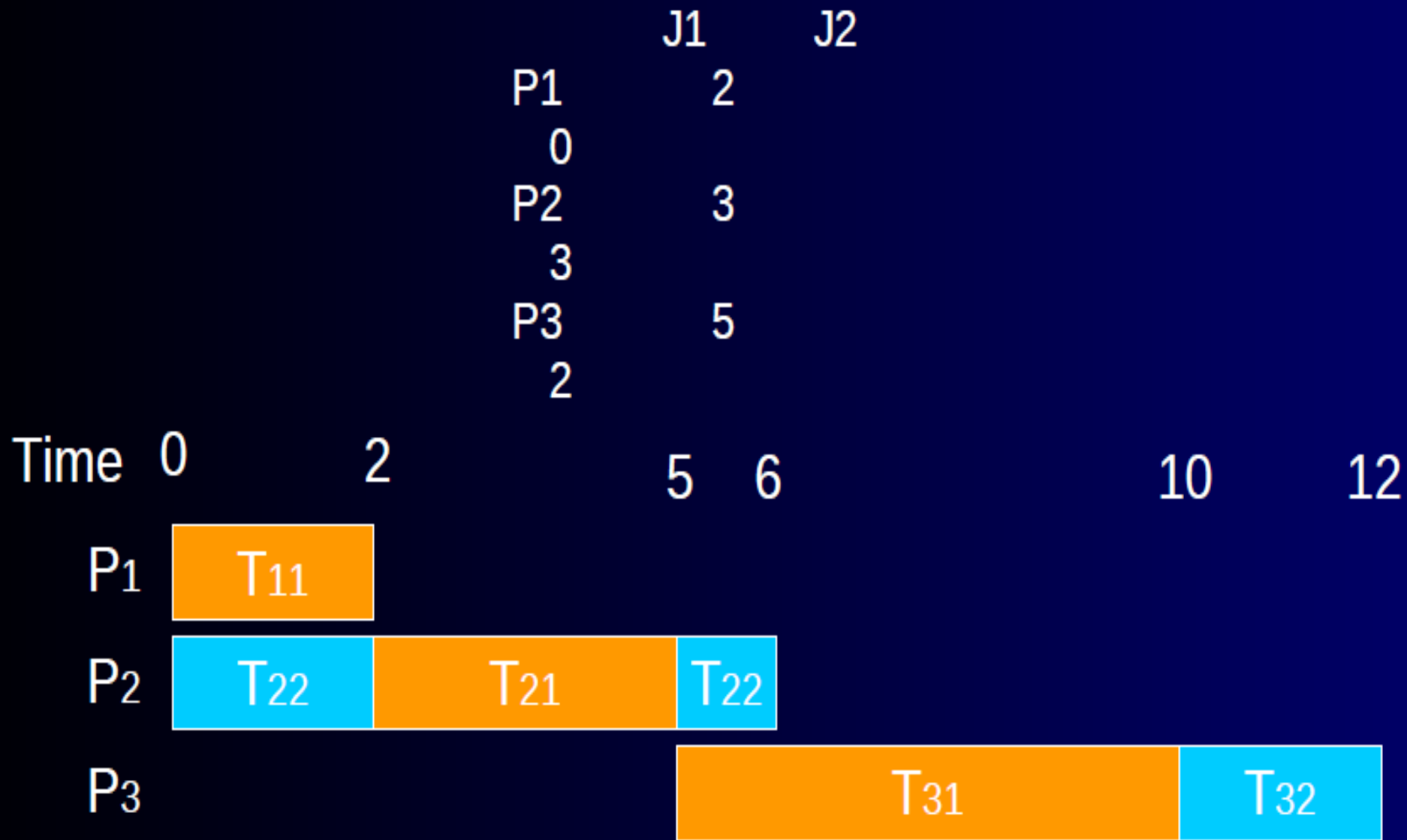
- Two jobs have to be scheduled on three processors.
- Task time is given by matrix

	Job1	Job 2
Processor 1	2	0
Processor 2	3	3
Processor 3	5	2

Schedule 1 (Non Preemptive)



Schedule 2(Preemptive)



Schedule

- Non-preemptive schedule is a schedule in which processing of task on any processor is not terminated until it is complete. (Schedule 1)
- A schedule in which it is not true is preemptive schedule. (Schedule 2)

Parameters

- Finish time $f_i(s)$ of job i is the time at which all tasks of job i have been completed in schedule s .
- Finish time $F(s)$ of the schedule s is given by
$$F(s) = \max \{ f_i(s) \}, \text{ amongst all jobs.}$$
- Mean flow time $MFT(s)$ is defined as
$$MFT(s) = 1/n \sum f_i(s)$$
- OFT (Optimal Finish Time) schedule
- POFT (Preemptive Optimal Finish Time) schedule

Flow Shop Scheduling

Example 5.28 Let $n = 4$, $(a_1, a_2, a_3, a_4) = (3, 4, 8, 10)$, and $(b_1, b_2, b_3, b_4) = (6, 2, 9, 15)$. The sorted sequence of a 's and b 's is $(b_2, a_1, a_2, b_1, a_3, b_3, a_4, b_4) = (2, 3, 4, 6, 8, 9, 10, 15)$. Let $\sigma_1, \sigma_2, \sigma_3$, and σ_4 be the optimal schedule. Since the smallest number is b_2 , we set $\sigma_4 = 2$. The next number is a_1 and we set $\sigma_1 = a_1$. The next smallest number is a_2 . Job 2 has already been scheduled. The next number is b_1 . Job 1 has already been scheduled. The next is a_3 and we set σ_3 . This leaves σ_2 free and job 4 unscheduled. Thus, $\sigma_2 = 4$. \square

Thank You