$x_{k+1} = f(x_k, u_k, w_k)$. w_k — r.v. takes only one value.

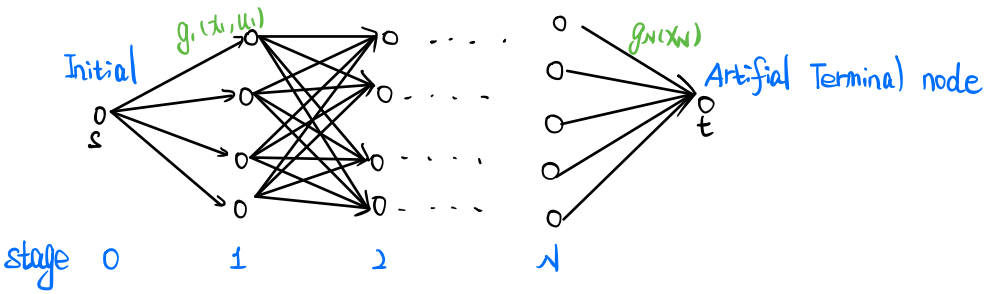
$\Rightarrow$ $x_{k+1} = f(x_k, u_k)$ $u_k = \mu_k(x_k)$ $k = 0, 1, \cdots, N-1$.

2.1. Finite - State System and Shortest Path.

State space S_k is finite for each k.

control u_k is associated with a transition from x_k to $x_{k+1} = f(x_k, u_k)$ at a cost $g_k(x_k, u_k)$

Finite - state deterministic problem can be represented a graph.



A path is a sequence of ones $(j_1, j_2), \cdots, (j_{k+1}, j_k)$, and length of the path is the sum of the length of its arcs.

finite state problem $\Leftrightarrow$ find a shortest path from s to t

Now denote

a_{ij}^k : cost of transition at stage k from $i \in S_k$ to $j \in S_{k+1}$.

a_{it}^N : terminal cost of state $i \in S_N$ $[g_N(i)]$

Then DP Algorithm takes the form :

$J_N(i) = a_{it}^N$ $i \in S_N$.

$$J_k(i) = \min_{j \in S_{k+1}} \{ a_{ij}^k + J_{k+1}(j) \}. \quad i \in S_k, \ k = 0, 1, \cdots, N-1.$$

Optimal Cost $J_0(s)$ = the length of the shortest path.

* A Forward DP Algorithm (Deterministic System).

$$\tilde{J}_N(j) = a_{sj}^0, \quad j \in S_1$$

$$\tilde{J}_k(j) = \min_{i \in S_{N-k}} \{ a_{ij}^{N-k} + \tilde{J}_{k+1}(i) \} \qquad j \in S_{N-k+1}, \ k = 1, \cdots, N-1.$$

* optimal cost. $\tilde{J}_0(t) = \min_{i \in S_N} \{ \tilde{J}_1(i) + a_{it}^N \}$.
* Forward Algorithm $\Leftrightarrow$ Backward Algorithm.

* Converting a Shortest Path Problem to a Deterministic Finite-State Problem.

* shortest path problem :

$\{1, \cdots, N, t\}$: set of nodes of a graph

a_{ij} : cost of moving from i to j.

t : special node. called destination.

$a_{ij} = \infty \Leftrightarrow$ no arc joining node i and j.

Target : find a shortest path from each node i to t.

Assumption : no circle is exist.

circle : $(i, j_1), (j_1, j_2), \cdots, (j_k, i)$; start and end is the same node.

We formulate the problem as one where we require exactly N moves but allow degenerate move from a node to itself with cost $a_{ii} = 0$

$J_k(i)$: optimal cost of getting from i to t in $N-k$ moves.

$\Rightarrow$ $J_0(i)$: cost of optimal path from i to t.

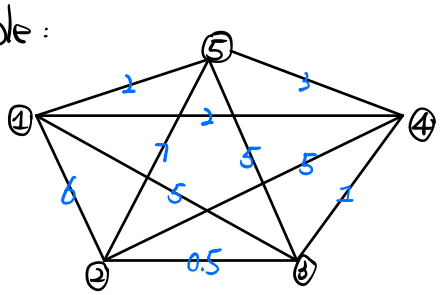
According to DP equation :

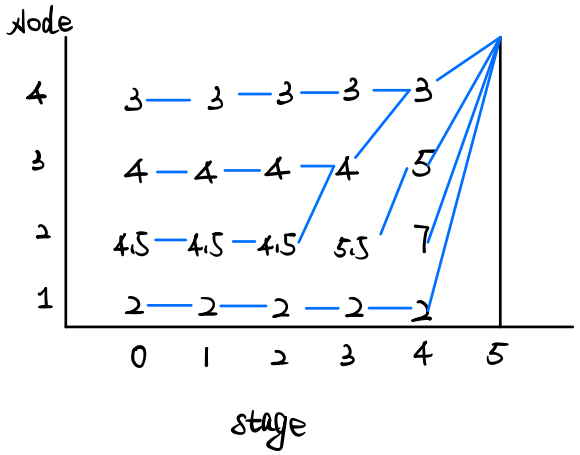
optimal cost from i to t in $N-k$ moves

$= \min_{j=1,\cdots,N} [a_{ij} + \text{optimal cost from } j \text{ to } t \text{ in } N-k-1 \text{ moves}]$

$\Rightarrow$ $J_k(i) = \min_{j=1,\cdots,N} [a_{ij} + J_{k+1}(j)]$ $\quad J_{N-1}(i) = a_{it}$, $i = 1, 2, \cdots, N$

Example :





Optimal path :

1 — 5

1 — 3 — 4 — 5

1 — 4 — 5

1 — 4 — 5

2.2 Some Shortest Path Applications.

* Critical Path Analysis.

A project requires the completion of several activities, and some activities must be done before the others. The duration of each activity is known in advance.

* TARGET: Find the time required to complete the project as well as the activities that cannot be delayed.

* critical activities: If we delay them then the whole project will delay

* Represent in Graph:

Nodes $1, \cdots, N$. : the phase of the project.

Arc (i,j). : an activity that starts once phase i is completed and has known the duration

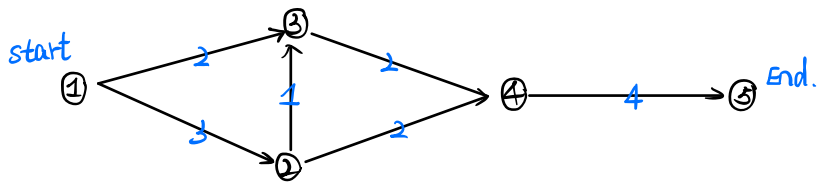
$t_{ij} > 0$

A phase j is complete : all activities (i,j) are completed.

Node 1: start of the project. (no coming arcs)

Node N: end of the project. (no outgoing arcs).

Important Characteristic: acyclic (have no cycle).



For any path $p = \{(1, j_1), \cdots (j_k, i)\}$ from node 1 to i:

D_p: the Duration of its activities.

$$D_p = t_{1j_1} + t_{j_2 j_3} + \cdots + t_{j_k, i}$$

T_i: time required to complete i.

$$T_i = \max_{\substack{\text{phase } p \\ \text{from 1 to } i}} D_p$$

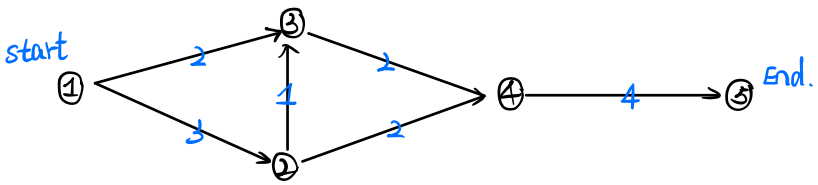
Find T_i $\Leftrightarrow$ Find the longest path from 1 to i

$\Leftrightarrow$ shortest path from 1 to i with cost $-t_{ij}$

S_i : the set of phases that do not depend on completion of any other phases.

$S_k = \{ i \mid$ all paths from 1 to i have k arcs or less$\}$.

state space for the DP problem



$S_0 = \{1\}$. $S_1 = \{1, 2\}$. $S_2 = \{1, 2, 3\}$. $S_3 = \{1, 2, 3, 4\}$. $S_4 = \{1, 2, 3, 4\}$.

$T_1 = 3$. $T_2 = 4$. $T_3 = 6$. $T_4 = 10$.

Critical path : 1 — 2 — 3 — 4 — 5.

2.3. Shortest Path Algorithm.

(*) use DP to solve shortest path problem

In particular, DP is preferred when the graph is acyclic.

(*) use shortest path problem to solve DP in special cases.

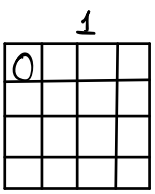
Example (The Four Queuens Problem).

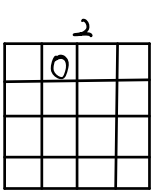
Requirements : * 4 Queuens must be placed on a 4x4 chessboard

* Each row each column. each diagonal contains at most 1 Queue

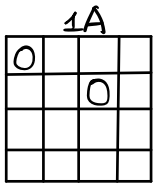
Solution :

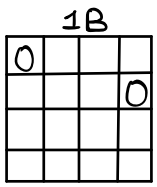
1)

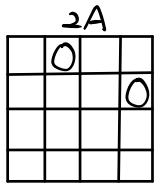




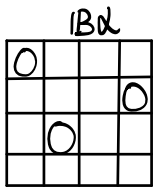
2)

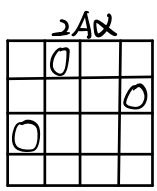




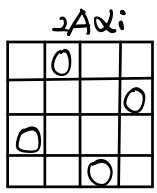


3)

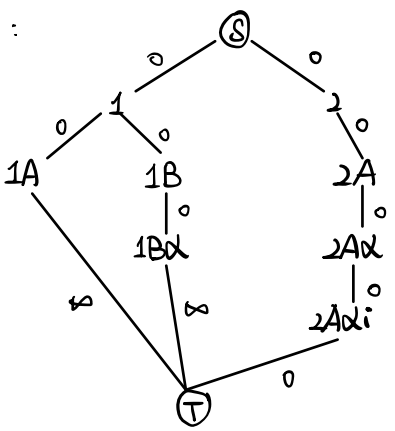




4)



graph :



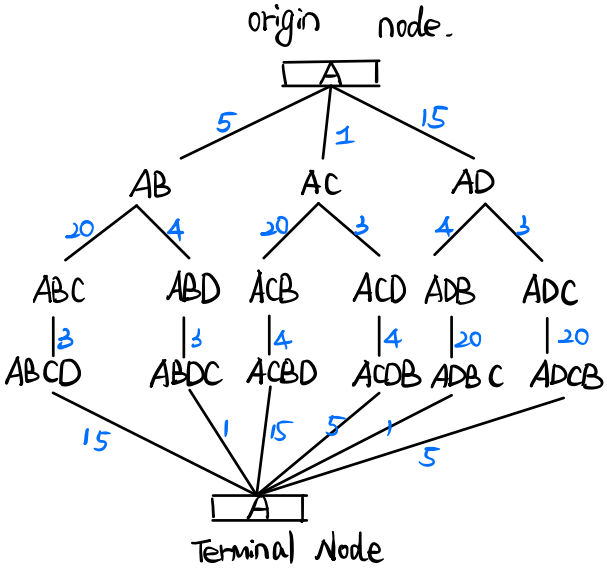
Example (Travelling Salesman Problem).

Question : N cities and the milage between each pair of cities is known. we need to find a minimum-milage trip that visits each of the cities exactly once and return to the origin node.

	A	B	C	D
A	0	5	1	15
B	5	0	20	4
C	1	20	0	3
D	15	4	3	0

origin A

Graph :



* Label Correcting Methods.

origin node : S

terminal node : T.

child node j of i : $\exists$ an arc (i,j) connecting i and j.

length of arc (i,j) : $a_{ij} \geq 0$. (no negative cycle).

TARGET : Find a shortest path from S to T

thoughts of algorithm : progressively discover shorter path from s to i

d_i : label of i.

* Each time d_i is reduced if find a shorter path from s to i
* If d_i is reduced. then we check to see if the labels d_j (child node of i) can be "corrected"
* that is correct from d_j to $d_i + a_{ij}$.

d_t : * maintained by a variable called UPPER.

d_s * remain 0 throughout the Algorithm.

* In initial stage. the labels $d_i = \infty$, $i \neq s$.

OPEN : the set of nodes which are currently active.

* active : candidates need further exploration by the Algorithm and possible inclusion in the shortest path.

* each node that has entered OPEN at least once , except s , is assigned "parent", which is some other node. Parent is used to trace the shortest path.

LABEL CORRECTING ALGORITHM.

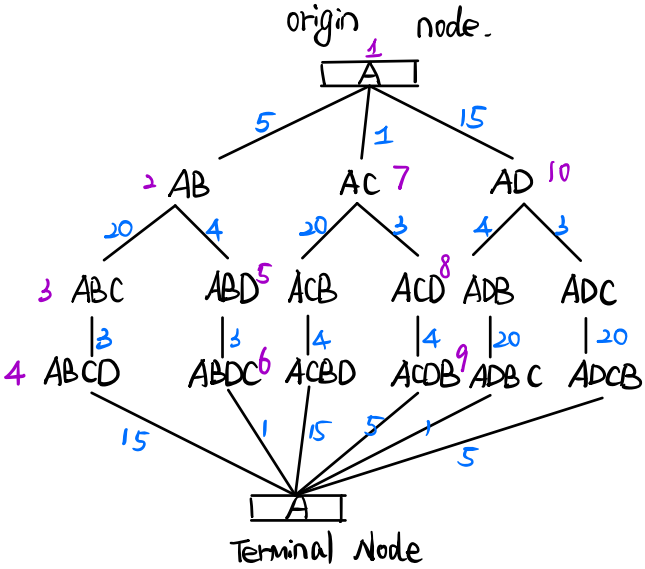
Step 1. Remove a node i from OPEN and for each child j of i go to step 2.

Step 2. If $d_i + a_{ij} < \min\{d_j, \text{UPPER}\}$. set d_j as $d_i + a_{ij}$ and set i to be the parent of j. In addition. $j \neq t$. place j in OPEN if it is not already in OPEN . if $j = t$. set UPPER to the new value. $d_i + a_{ij}$ of d_t.

Step 3. If OPEN is empty . terminate.
else go to step 1.

Proposition 2.3.1.

If there exist at least 1 path from the origin to the destination. the label correcting algorithm terminates with UPPER = the shortest distance from s to t. Otherwise, algorithm returns ∞.



Iter No.	Node exiting OPEN	OPEN	UPPER.
0	—	1	∞
1	1	2, 7, 10	∞
2.	2	3 5 7 10	∞
3	3	4 5 7 10	∞
4	4	5 7 10	43
5	5	6 7 10	43
6	6	7 10	13
7	7	8, 10	13
8	8	9. 10	13
9	9	10	13
10	10	—	13.