# CS2020 – Data Structures and Algorithms Accelerated

# Lecture 20 – DP on General Graph

stevenhalim@gmail.com

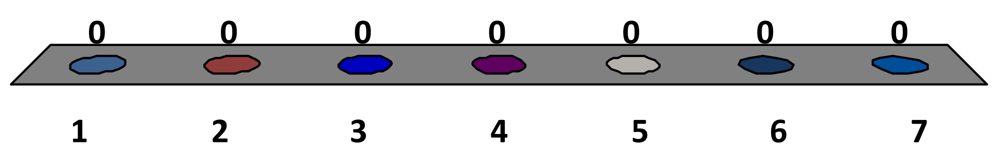
National University of Singapore
School of Computing

# Outline

- What are we going to learn in this lecture?
  - Review + PS9 Reminder + PS10 Preview
  - DP on General Graph
    - Is it possible to write a recurrence on graph that contains cycle?
    - Well-known graph algorithms versus DP?
    - The key point of this lecture: Conversion to a DAG
    - The classical Traveling Salesman Problem (TSP)

# What is the LIS of X = {8, 3, 6, 4, 5, 7, 7}?

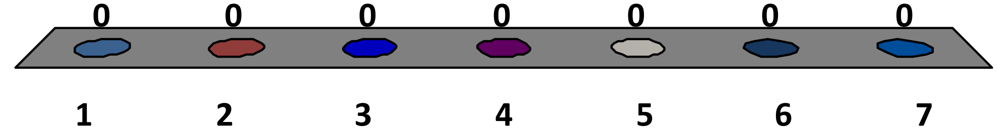
1. 1
2. 2
3. 3
4. 4
5. 5
6. 6
7. 7




0 of 54

# What is the LNDS of X = {8, 3, 6, 4, 5, 7, 7}?
# ND = Non Decreasing

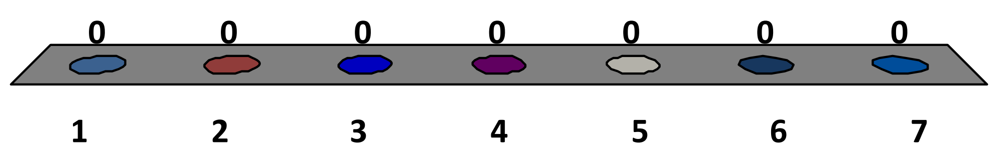
1. 1
2. 2
3. 3
4. 4
5. 5
6. 6
7. 7




0 of 54

# What is the LDS of X = {8, 3, 6, 4, 5, 7, 7}?
# D = Decreasing

1. 1
2. 2
3. 3
4. 4
5. 5
6. 6
7. 7

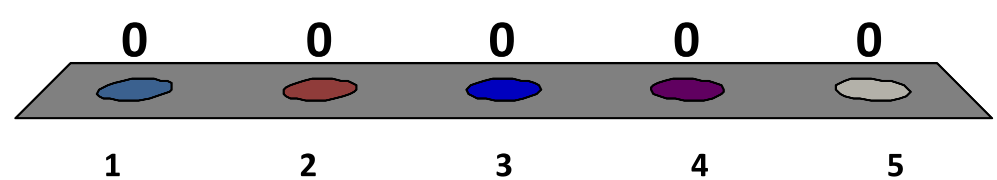



**0 of 54**

How many paths to go from (0, 0) to (3, 3) if you can only go **down** or **right** at every cell?

1. 5
2. 10
3. 20
4. 40
5. $\propto$

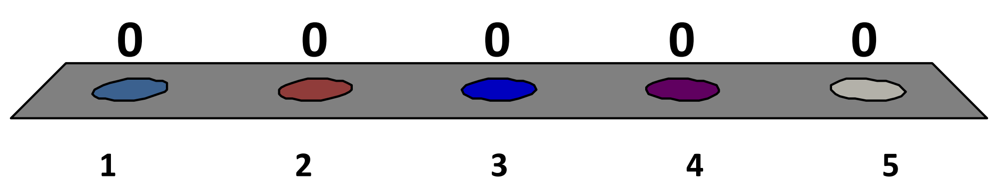
| | | | |
|---|---|---|---|
| (0, 0) | (0, 1) | (0, 2) | (0, 3) |
| (1, 0) | (1, 1) | (1, 2) | (1, 3) |
| (2, 0) | (2, 1) | (2, 2) | (2, 3) |
| (3, 0) | (3, 1) | (3, 2) | (3, 3) |




**0 of 54**

You have to solve an SSSP problem on weighted graph (+/-) with just $V < 20$ vertices, you will use

1. Bellman Ford’s
2. Dijkstra’s (original)
3. Dijkstra’s (modified)
4. BFS
5. Floyd Warshall’s



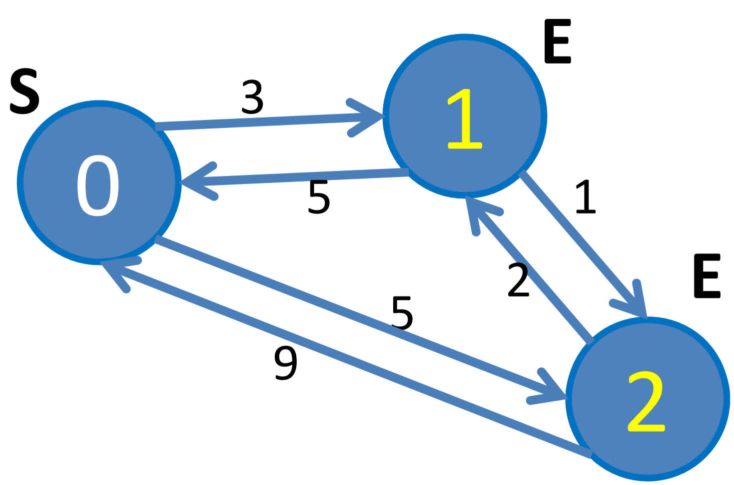

0 of 54

# PS9 Reminder + PS10 Preview

- PS9
  - Deadline is tomorrow, Wed 6 April 2011, 2pm
  - Minor sample test data error in “life.java”
  - Space for announcements/bug fixes, etc
- PS10 (just opened)
  - The last PS that gives you the largest EXP points, yippie ☺
  - There are two (ehem…) programming tasks
    - One is DP on general graph converted to DAG (as discussed today)
    - One is DP problem which should be easier if not viewed as a graph problem (will be discussed this coming Friday)
    - Both are from a recent programming competition for Singapore high school students held on 5 March 2011, the NOI 2011

# Motivating Problem

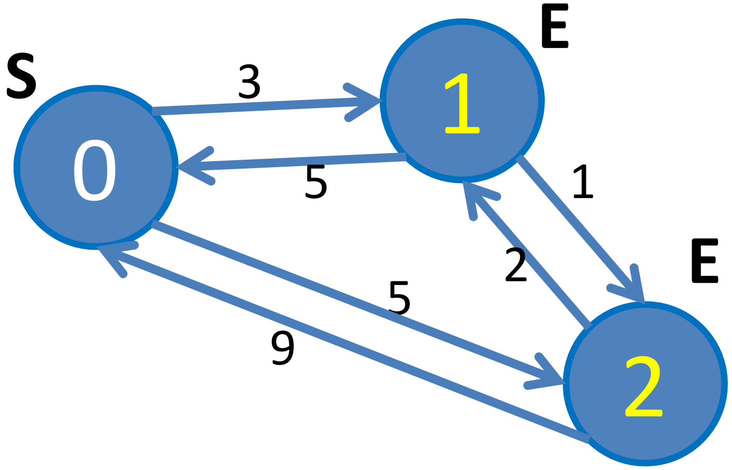
- UVa 10702 – Traveling Salesman
  - There are **C** cities; A salesman starts his sales tour from city **S** and can end his sales tour at any city labeled with **E**
  - He wants to visit many cities to sell his goods; Every time he goes from city **U** to city **V**, he obtains **profit[U][V]**
    - profit[U][U] is always 0
  - What is the maximum profit that he can get?



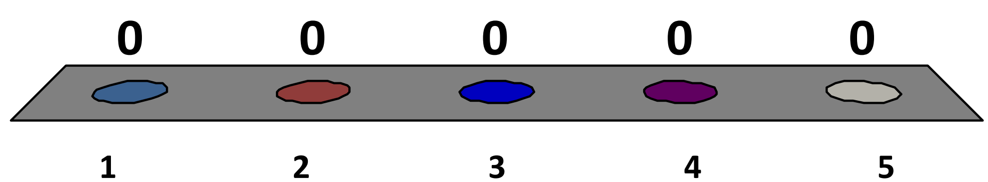

profit[U][V] is shown as the weight of edge(U, V)

# What is the maximum profit that he can get?

1. 3
2. 5
3. 7
4. 17
5. ∞

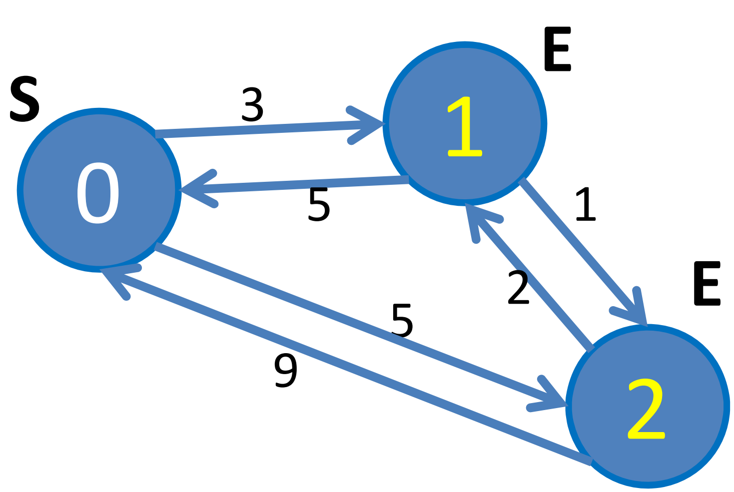



0 of 54



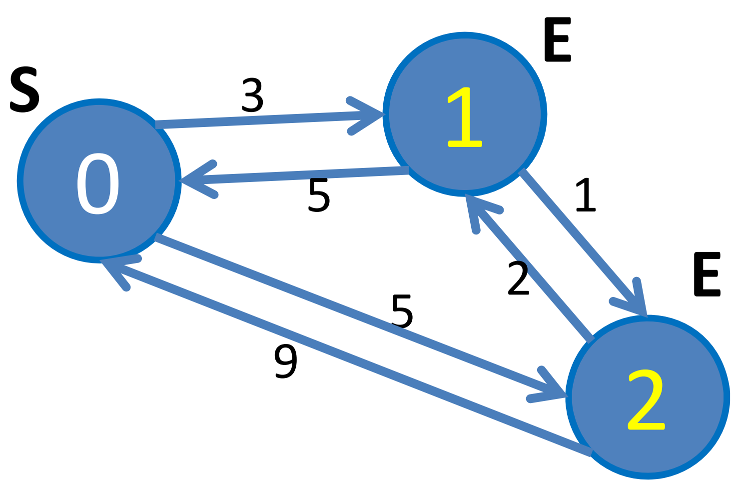
# Reducing to SS Longest (non simple) Path Problem but on General Graph

- This is a problem of finding the **longest (non simple) path** on **general graph** :O
  - General graph has something that does not exist in DAG discussed earlier in Lecture18: **cycle(s)**
  - There are several **positive** weight cycles in this graph, e.g. 0→1→2→3→0 (weight 13); 0→1→0 (weight 8), etc
    - The salesman can keep re-visiting these cycles to get more $$ :O



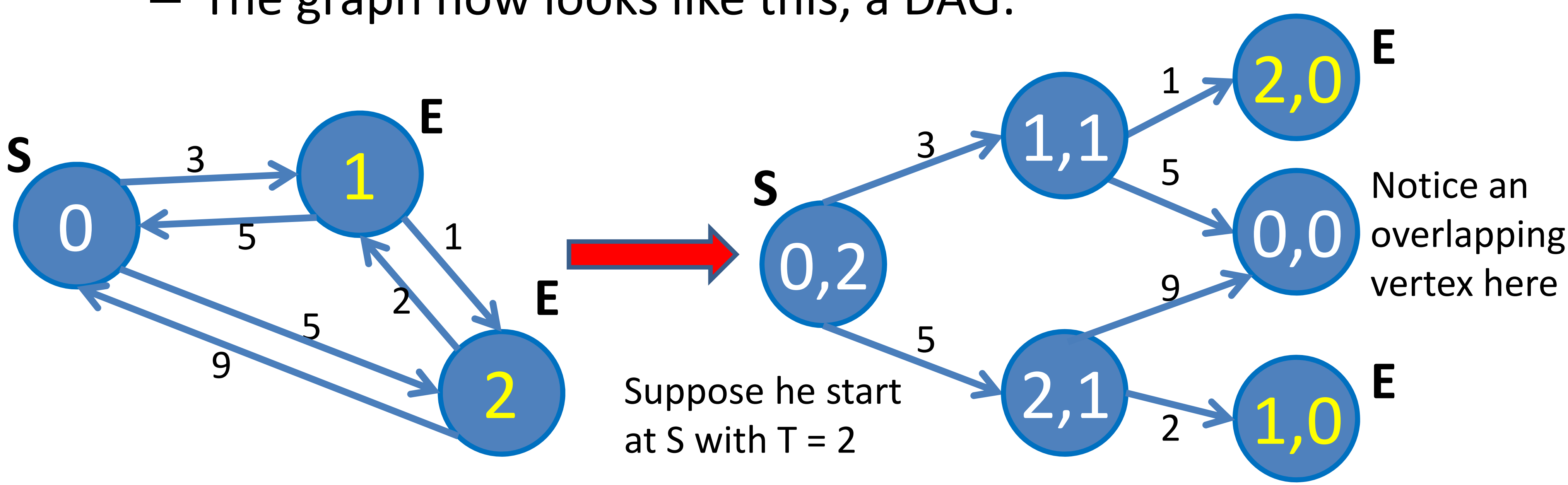
# A Note About Longest Path Problem on General Graph

- The longest **(non simple)** path from S = 0 to any of the E will have ∝ weight
  - One can go through any **positive weight cycle** to obtain ∝
- The longest **(simple)** path from S = 0 to any of the E is: 0→2→1 with weight 5+2 = 7
  - But this is hard, as discussed in DG8 and shown again later
  - And we cannot write a recurrence if the graph is cyclic



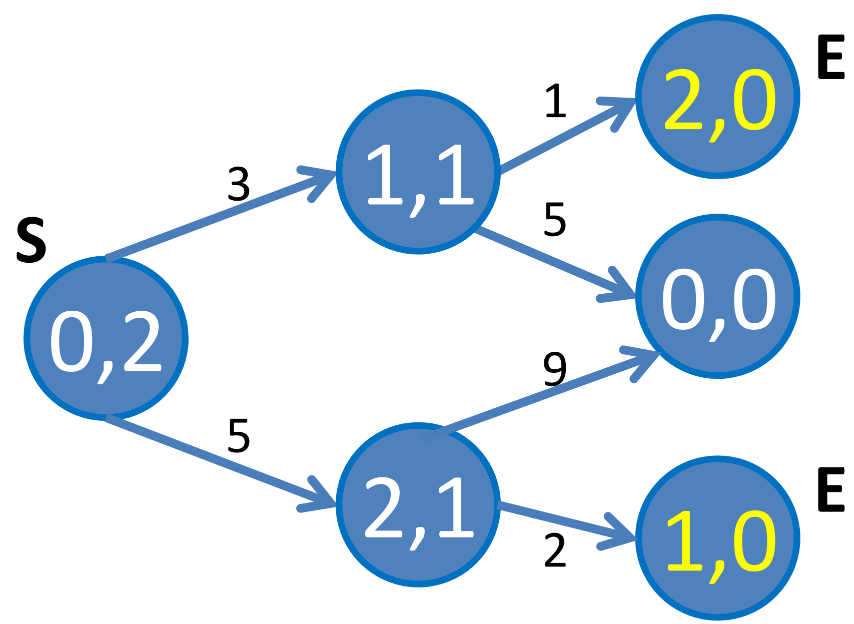
# Conversion to a DAG

- The actual problem (UVa 10702) has an extra parameter that convert the general graph to a DAG
  - The salesman can only make T inter-city travels :O
  - In a valid tour, he must arrive at an ending vertex after T step
  - Now each vertex has additional parameter: num of steps left
  - The graph now looks like this, a DAG:



# A Note About Longest Path Problem on Directed Acyclic Graph

- There is no longest *non simple* path on DAG
- This is because every paths on DAG are simple, including the longest path ☺
- So we can use the term SSLP on DAG, but we have to use the term SSL(simple)P on general graph



# Graph versus DP

- What is the solution for the SSLP on DAG problem?
  - We are already familiar with this (from Lecture18)
  - SS Longest paths on DAG can be solved with either:
    - Find topological order and “stretch” edges according to this order
    - Or write a recursive function with memoization
- But this is *harder* to be solved as a graph problem
  - The vertices now contain pair of information:
    - Vertex number and number of steps left
  - The number of vertices is not V, but now V*T
  - Graph implementation is going to be more difficult…

# DP Solution (1)

- Let's solve this problem with Dynamic Programming
- Let **get_profit(u, t_left)** be the maximum profit that the salesman can get when he is at city **u** with **t_left** number of steps to go:
  - if t_left = 0
    - If the salesman can end his tour at city u, i.e. city u has label E;
      - Then get_profit(u, t_left) = 0
    - else if the salesman cannot end his tour at city u;
      - Then get_profit(u, t_left) = -INF (a bad choice)
  - else,
    - get_profit(u, t_left) = max(profit[u][v] + get_profit(v, t_left – 1)) for all v $\in$ [0 .. C - 1]

# DP Solution (2)

- In Java code (see UVa10702.java):

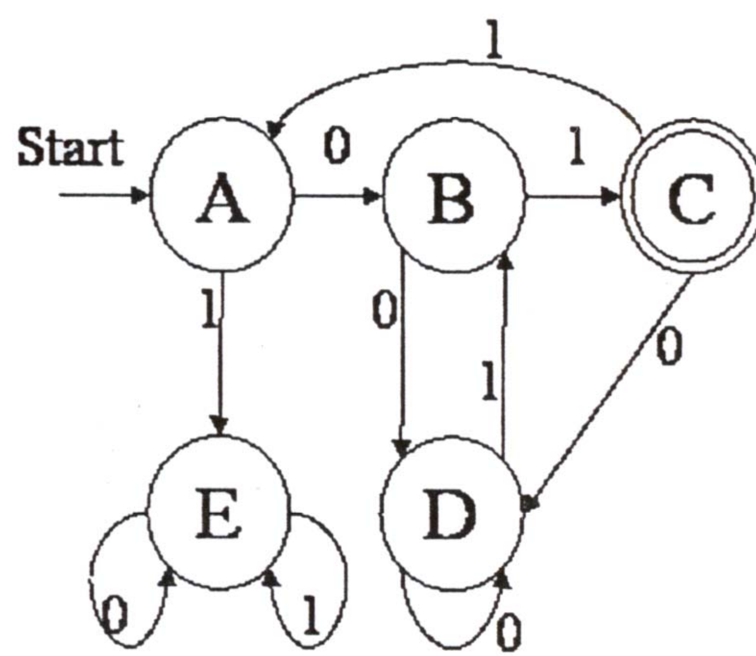
```
private static int get_profit(int u, int t_left) {
  if (t_left == 0) // last inter-city travel?
    return canEnd[u] ? 0 : -INF;
  if (memo[u][t_left] != -1) // computed before?
    return memo[u][t_left];

  memo[u][t_left] = -INF;
  for (int v = 1; v <= C; v++)
    memo[u][t_left] = Math.max(memo[u][t_left],
       profit[u][v] + get_profit(v, t_left - 1));
  return memo[u][t_left];
}
```

# DP Analysis

- What is the num of distinct states/space complexity?
  - That is, the vertices in the DAG
    - Answer: O(C*T)
- What is the time to compute one distinct state?
  - That is, the out-degree of a vertex
    - Answer: O(C), actually C - 1,but it is O(C)
- What is the overall time complexity?
  - That is, the total number of edges in the DAG
  - This is number of vertices * out degree of each vertex
    - Or number of distinct states * time to compute one distinct state
    - Answer: $O((C*T) * C) = O(C^2*T)$

# Another Problem

- UVa 910 – TV Game
  - There are **N** vertices (up to 26 vertices; labeled 'A' to 'Z')
    - Some of them are special (drawn with double circle)
  - Each vertex has two outgoing edges
    - One that has label 0 and one that has label 1
    - The edge may be a self loop :O
      - Not a simple graph…
      - This is a multi graph, ugh…
  - Question: How many ways to reach the special vertices from vertex A in *exactly* **m** moves ($0 \le m \le 30$)?

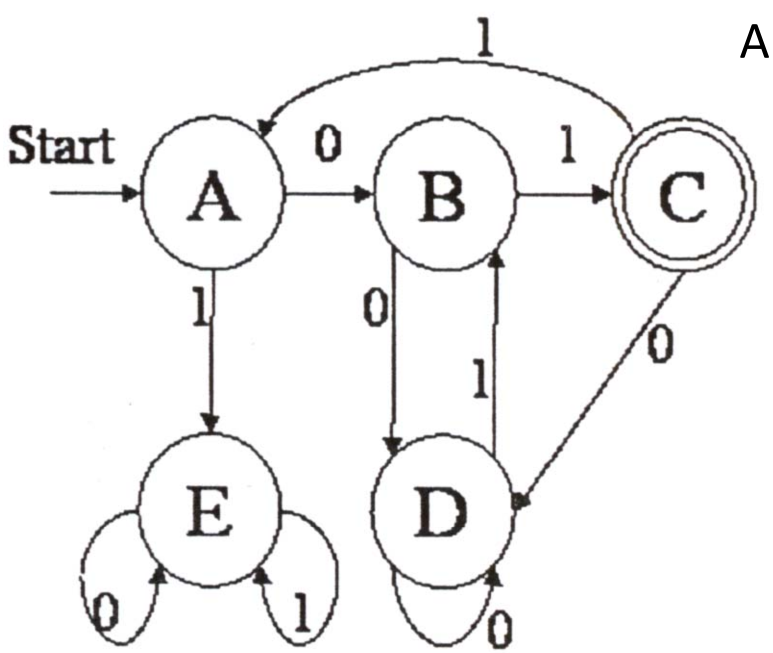


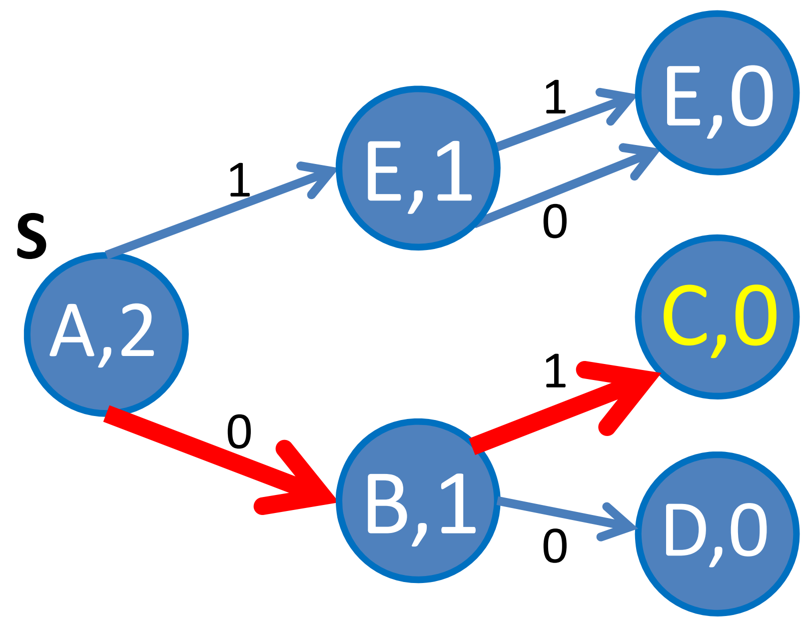

C is a special vertex

# Conversion to a DAG

- If we do not convert the multi graph into a DAG first, we may end up in infinite loop, like A→E→E →E …
  - Originally = Counting Paths in Multi Graph
  - After conversion = Counting Paths in DAG ☺

DAG for **m = 2,** each vertex has label (vertex_ID, m_left)
Answer = only one path = (A, 2) → (B, 1) → (C, 0)



# DP Solution (1)

- We will not bother with topological sort (graph way)
- Let **ways(u, m_left)** be the number of ways to reach any special vertex from vertex **u** with **m_left** number of moves to go:
  - if m_left = 0
    - if vertex u is special
      - Then ways(u, m_left) = 1, we found one way
    - else if vertex u is not special
      - Then ways(u, m_left) = 0, do not count this
  - else, combine the ways from either taking edge 0 or 1
    - ways(u, m_left) = ways(if0[u], m_left - 1) + ways(if1[u], m_left - 1)
      - if0[u] tells the next vertex from u if edge with label 0 is chosen
      - if1[u] tells the next vertex from u if edge with label 1 is chosen

# DP Solution (2)

- In Java code (see UVa910.java):

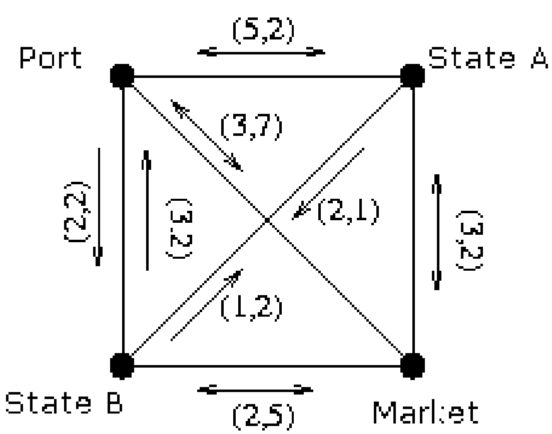
```
private static int ways(int u, int m_left) {
  if (m_left == 0) // no more move left?
    return special[u] ? 1 : 0;
  if (memo[u][m_left] != -1) // computed before?
    return memo[u][m_left];
  // two options, take edge 0 or edge 1
  // if0 and if1 is another "graph data structure"
  return memo[u][m_left] = ways(if0[u], m_left - 1) +
                           ways(if1[u], m_left - 1);
}
```

# DP Analysis

- What is the num of distinct states/space complexity?
  - Answer: O(N*m)
- What is the time to compute one distinct state?
  - Answer: O(1), always two out-going edges per vertex
- What is the overall time complexity?
  - Answer: O((N*M) * 1) = O(N*M)

# One More Problem (for DG9)

- SPOJ 101 – FISHMONGER
  - Given two **n x n** matrices (3 ≤ **n** ≤ 50)
    - One gives travel time between two cities
    - The other gives toll between two cities
  - Also, given an available time **t** (1 ≤ **t** ≤ 1000)
  - Find a route from source (city **0**) so that the fishmonger arrives at destination (city **n - 1**) within a certain time **t**
    - And this route must be the one with the minimum toll cost



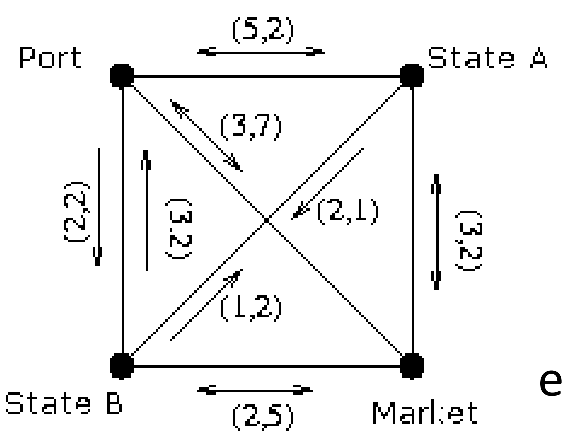

edge weight = (time, toll)

if **t = 7**, then

a. Direct path = Port → Market
   uses 3 units of time, toll cost = 7 (not optimal)
b. Path = Port → State B → State A→ Market
   uses 6 units of time, toll cost = 6 (optimal)

# Is This an SSSP Problem?

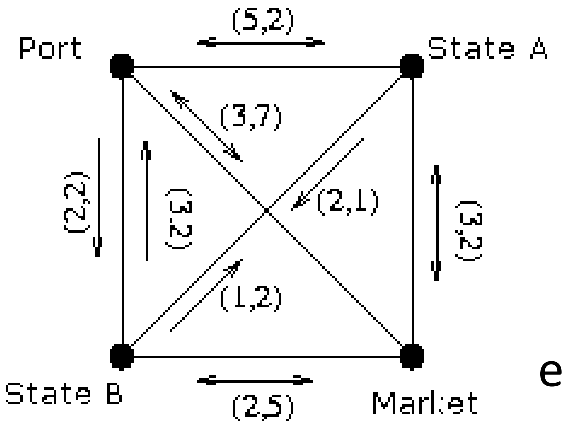
- To be discussed in DG9 ☺




edge weight = (time, toll)

# DP Solution + Analysis

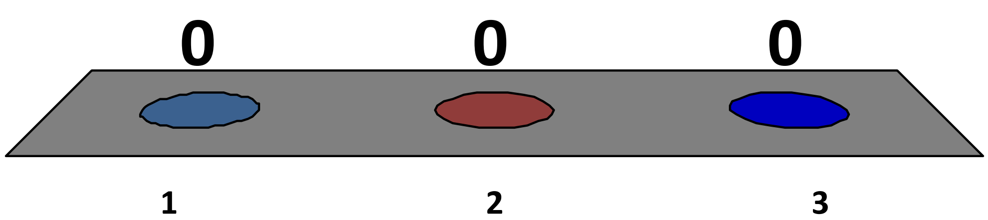
- To be discussed in DG9 ☺




edge weight = (time, toll)

# So far…

1. I am OK with DP techniques ☺
2. I can understand most concepts although some (minor) details are still not clear
3. Scary…, I have been really lost since the first topic of DP (Lecture18-now ☹), I need help




0 of 54

5 minutes break

Then, another example of DP on General Graph

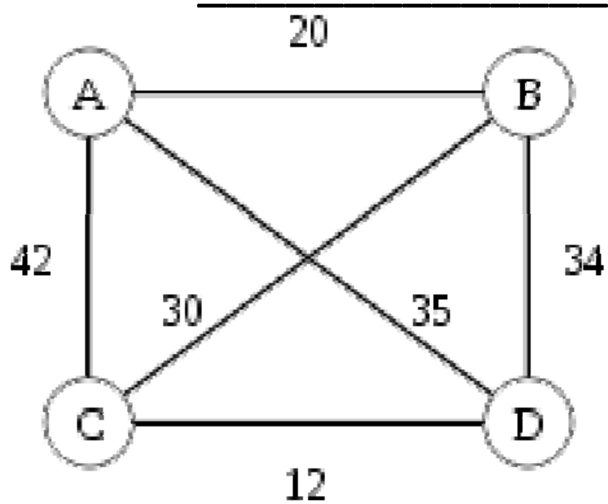
# TRAVELING SALESMAN PROBLEM

# Traveling Salesman Problem (TSP)

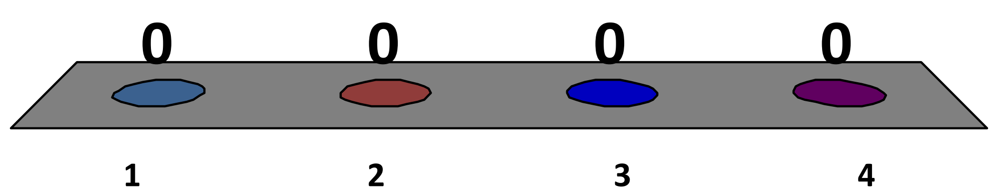
- The TSP is actually “simple” to describe:
  - Given a list of V cities and their pairwise distances
    - That is, a **complete weighted graph**
    - This is a general graph
  - Find a shortest tour that visits each city **exactly once**
    - Thus the tour will have exactly V edges, a **simple** tour
    - The tour must start and end at the same city
- Note that this problem is different from UVa 10702 shown at the beginning of this lecture
  - Take some time to examine the differences

# The shortest tour for this TSP instance is…
## (you will need some time to compute this)

1. Tour A-B-D-C-A with cost ______________________________
2. Tour A-B-C-D-A with cost ______________________________
3. Tour A-D-B-C-A with cost ______________________________
4. Other tour ___________________ with cost ____________________

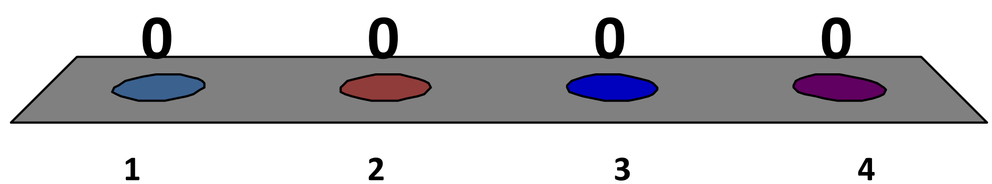



**0 of 54**



# How many possible tours are there in a TSP instance of V cities?

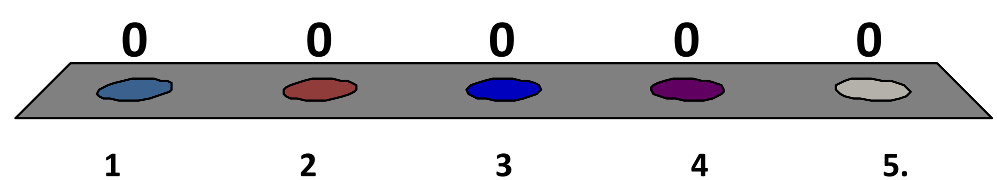
1. V valid tours
2. $V^2$ valid tours
3. V log V valid tours
4. V! valid tours




**0 of 54**

# What is the value of “10!” ?

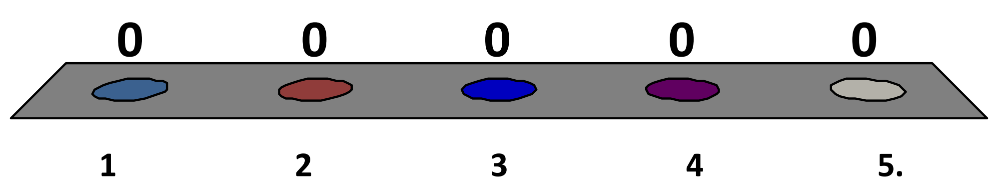
1. 10
2. 100
3. 3628800
4. 10000000
5. 9.332621544394415
   2681699238856267
   e+157




**0 of 54**

# What is the value of “100!” ?

1. 10
2. 100
3. 3628800
4. 10000000
5. 9.3326215443944152681699238856267e+157




0 of 54

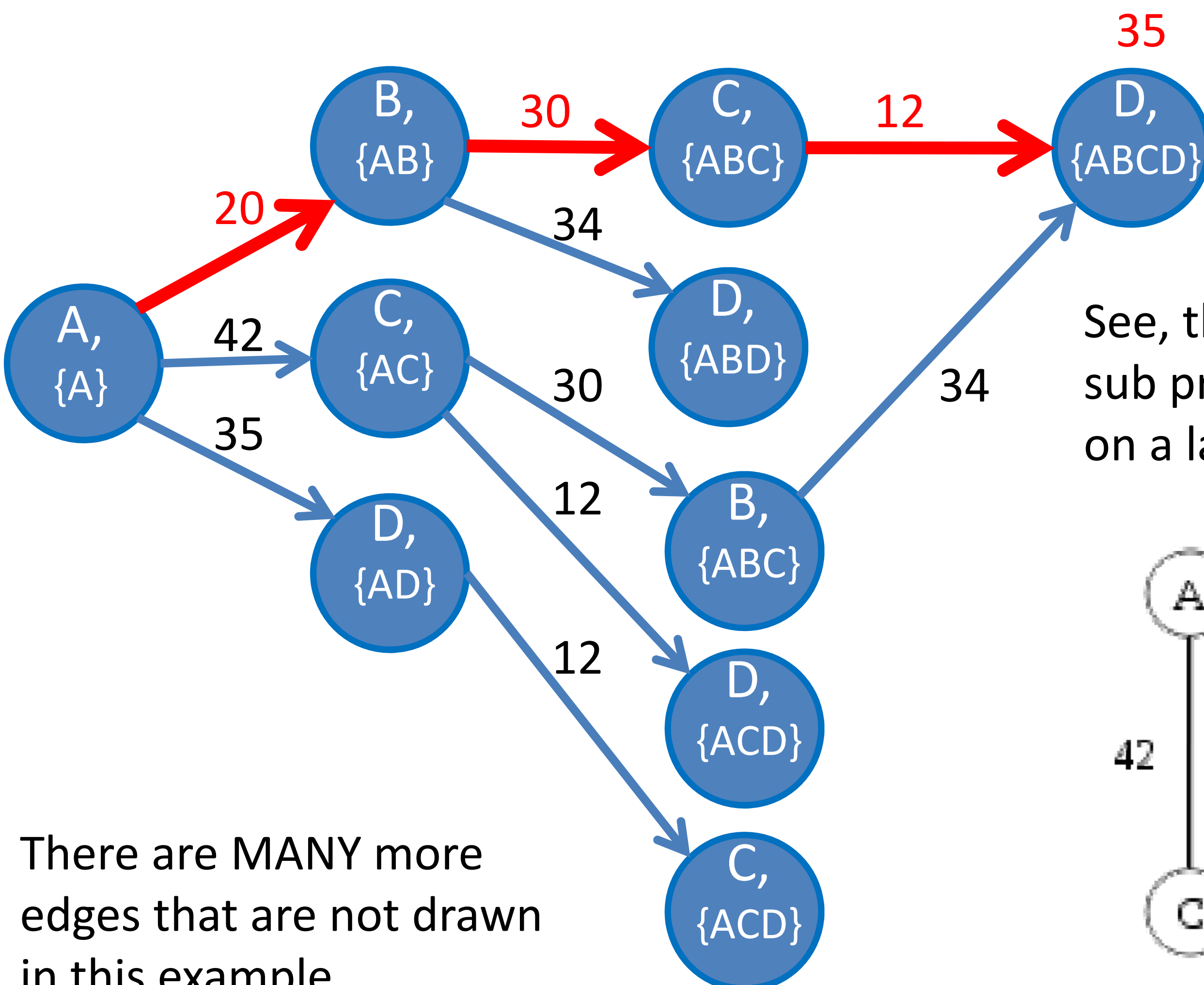
# Brute Force (Naïve) Solution

- In sketch:
  - Try all V! tour permutations
  - Pick the one with the minimum cost…
- This sketch is too coarse for proper implementation
  - Let's analyze TSPDemo.java
  - I start from DFSrec (Lect14), $O(V + E)$, $V = N$, $E = N^2 \sim O(V^2)$
  - I will show how to change DFSrec into backtracking routine that tries all permutations, while still using the **visited** flag
  - This is a $O(V!)$ algorithm, as there are V! possible tours
  - Then, I will show that using DP at this point is not correct…

# Converting to a DAG (1)

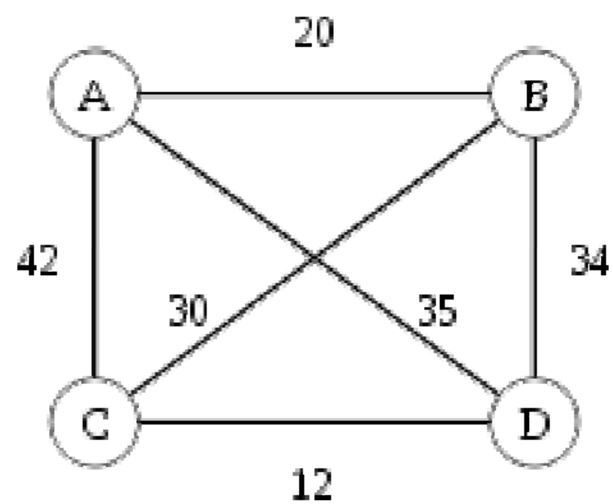
- To do backtracking in this complete general graph, we have to use the **visited** flag that is turned on when entering the recursion **and turned off when exiting the recursion (different from DFSrec)**
  - Some of you are trying to do this in Quiz 2…
- This essentially converts the complete general graph into a DAG, where each vertex is now has one more parameter: the set of vertices already visited up to the current one, see the next slide for a figure

# Converting to a DAG (2)




The path in red (it is a tour actually) has the minimum total weight of 97

See, there are (lots) of overlapping sub problem, try a drawing TSP DAG on a larger instance, say n = 10…




There are MANY more edges that are not drawn in this example

# DP Solution (1)

- Now, how many vertices are there in this DAG?
  - $N * 2^N$
  - Because we can reach a certain vertex with $2^N$ possible visited vertices (including this vertex)
- Then, how to store this “set of boolean” effectively?
  - Subset technique: bitmask
  - New “data structure” for lightweight set of boolean

# New DS: lightweight set of boolean

- An integer is stored in binary in computer memory
  - int x = $7_{10}$ (decimal) is actualy $111_2$ (binary)
  - int y = $12_{10}$ is $1100_2$
  - int z = $83_{10}$ is $1010011_2$
- We can use this sequence of 0s and 1s to represent a small set of boolean
- N-bits integer can represent N objects
  - 32 objects for a 32-bit integer
  - If i-th bit is 1, we say object i is in the set/active/visited Otherwise, object i is not in the set/not active/not visited

# Bit Operations (1)

- To check whether bit **i** is on or off
  - **x & (1 << i)**
  - Example:
    - x = $25_{10}$ ($11001_2$), check if bit 2 (from right, 0-based indexing) is on
    - x & (1 << 2) = 25 & 4

    11001

    00100

    -------- & (bitwise AND operation)

    00000

    - x = $0_{10}$ = ($00000_2$) now, that means bit i = 2 (from right) is **off**

# Bit Operations (2)

- To check whether bit **i** is on or off
  - **x & (1 << i)**
  - Example:
    - x = $25_{10}$ ($11001_2$), check if bit 3 (from right, 0-based indexing) is on
    - x  & (1 << 3) = 25 & 8

    11001

    01000

    -------- & (bitwise AND operation)

    01000

    - x = $8_{10}$ = ($01000_2$) now, that means bit i = 3 (from right) is **on**

# Bit Operations (3)

- To turn on bit **i** of an integer **x**
  - **x | (1 << i)**
  - Example:
    - x = $25_{10}$ ($11001_2$), turn on bit 2 (from right, 0-based indexing)
    - x | (1 << 2) = 25 | 4 =

    11001

    00100

    -------- | (bitwise OR operation)

    11101

    - x = $29_{10}$ = ($11101_2$) now, now bit 2 (from right) is on

# Bit Operations (4)

- To turn on bit **i** of an integer **x**
  - **x | (1 << i)**
  - Example:
    - x = $25_{10}$ ($11001_2$), turn on bit 3 (from right, 0-based indexing)
    - x | (1 << 3) = 25 | 8 =

    11001

    01000

    -------- | (bitwise OR operation)

    11001

    - x = $25_{10}$ = ($11001_2$) now, no change if bit 3 (from right) is already on

# DP Solution (2)

```
private static int[][] memo2 = new int[16][1 << 16];
// 1 << 16 = 2^16

private static int DP_TSP(int u, int vis) {
  if (vis == (1 << N) - 1) // all vertices have been visited
    return AdjMatrix[u][0]; // no choice, return to vertex 0
  if (memo2[u][vis] != -1) // this is correct
    return memo2[u][vis];

  int bestAns = INF;
  for (int v = 0; v < N; v++)
    if (AdjMatrix[u][v] > 0 && (vis & (1 << v)) == 0)
      bestAns = Math.min(bestAns,
                AdjMatrix[u][v] + DP_TSP(v, (vis | (1 << v))));
    memo2[u][vis] = bestAns;
  return bestAns;
}
```

# DP Analysis

- What is the num of distinct states/space complexity?
  - Answer: $O(N*2^N)$
- What is the time to compute one distinct state?
  - Answer: $O(N)$, must check all neighbors of a vertex as this is a complete graph, each vertex has out-degree N
- What is the overall time complexity?
  - Answer: $O((N*2^N) * N) = O(N^2*2^N)$

# Summary

- By definition, a general graph has cycles
  - You cannot write a recursive formula for a cyclic structure…
  - Therefore we cannot use DP technique on general graph :O, err??
- In this lecture, we have seen how to convert general graphs into DAGs by introducing (one) extra parameter
  - Now we can write recursive formulas and use DP
  - We can analyze space and time complexity of DP solution easily
- In the next lecture, Lecture21, we will see the true form of DP
  - We will see example problems that are "inappropriate" to be viewed as graph problems, it can have "several" parameters…
  - This is the last examinable topic of CS2020
    - And also possibly one of the hardest…