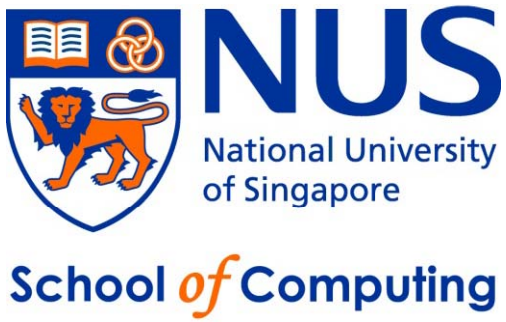
CS2010 – Data Structures and Algorithms II

Lecture 10 – Algorithms on (Implicit) DAG

stevenhalim@gmail.com



Admins

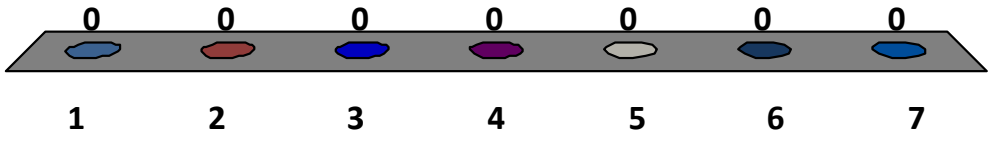
- We will continue from last week's "Counting Paths on DAG" first
- Reminder: Quiz 2 this Saturday, 27 October 2012 from 10am to 12pm (2 hours) @ COM1-2-SR1
  - It will be **harder** and **longer** than Quiz 1
  - Give Section 5 (32+3 marks) **(much) more than 15 minutes** as you will need at least 15 minutes to read and understand what is being asked, before you can answer some guiding questions and get at least some marks from this section
  - The final solution requires one "aha" moment not previously taught in class :O…

Outline

- What are we going to learn in this lecture?
  - DP Algorithms on (Implicit) DAG
    - Is it possible to write a recurrence for a graph problem when the graph has cycle(s)?
    - What if the problem has 'simplifying assumption(s)' that makes it easier?
    - The key point of this lecture:
      - Conversion of a general graph into a DAG with an addition of just one extra parameter
        (the harder versions are reserved for CS3230? or CS3233)
        - » Two simpler examples
        - » One harder example: The classic TSP DP solution
    - Reference: CP2.5 Section 3.5 + 4.7.1

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

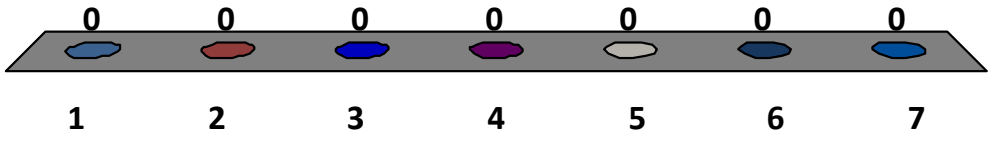
1. 1
2. 2
3. 3
4. 4
5. 5
6. 6
7. 7



What is the L**ND**S 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

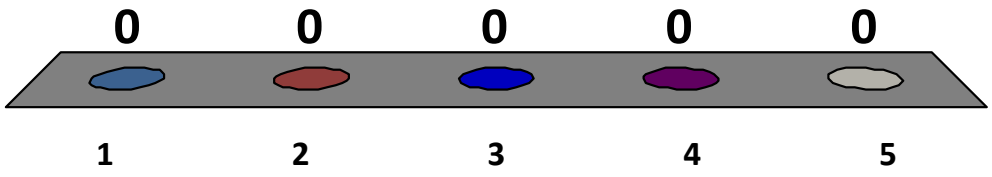


How many distinct paths to go from (0, 0) to (3, 3) if you can only go **down** or **right** at every cell?

1. 6
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 120

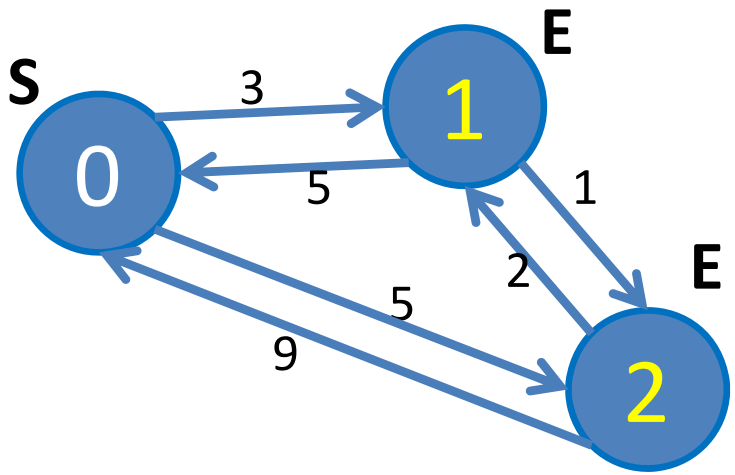


UVa 10702 – Traveling Salesman

EXAMPLE 1

Motivating Problem

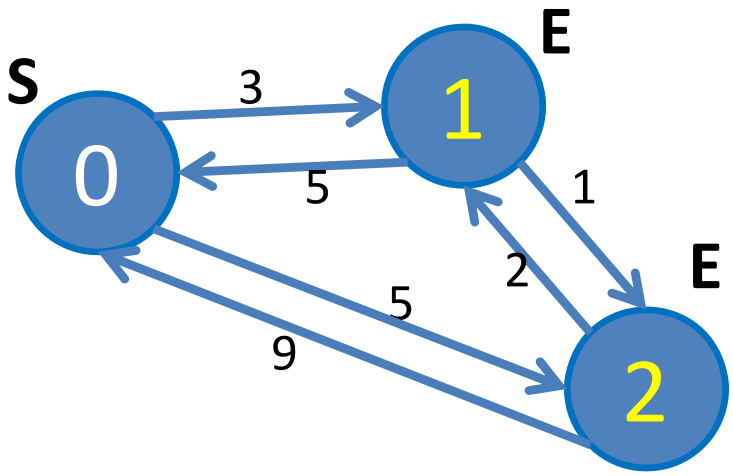
- UVa 10702 – Traveling Salesman
  - There are **C** cities; A salesman starts his sales tour from city **S** and can end his sales tour at any city labeled with **E**
  - He wants to visit many cities to sell his goods; Every time he goes from city **U** to city **V**, he obtains **profit[U][V]**
 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

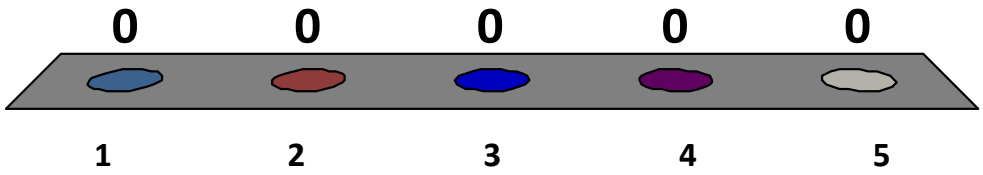


profit[U][V] is shown as the weight of edge(U, V)

What is the maximum profit that he can get?

1. 3
2. 5
3. 7
4. 17
5. ∞

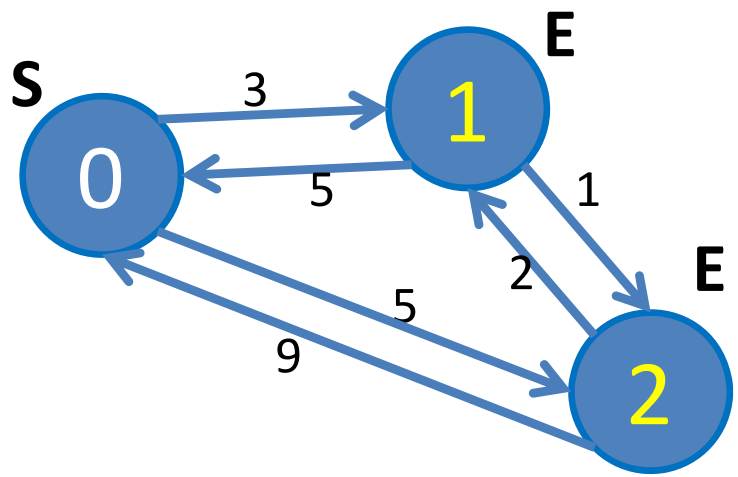




0 of 120

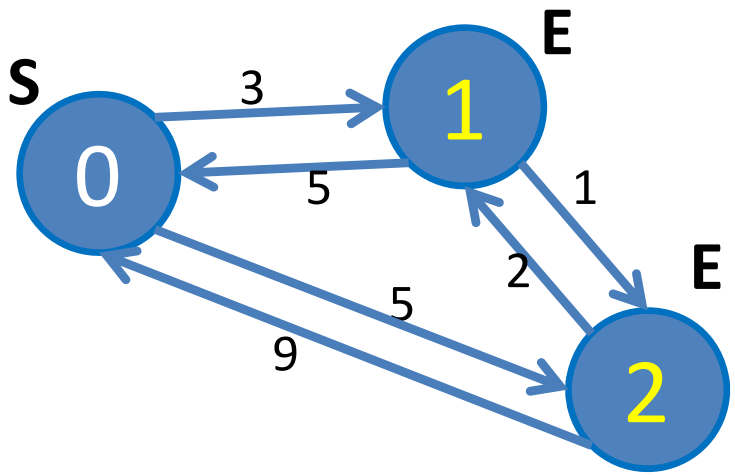
Reducing to SS Longest (non simple) Path Problem but on General Graph

- 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 in previous lecture: **Cycle(s)**
  - There are several **positive** weight cycles in this graph, e.g. 0→1→2→0 (weight 13); 0→1→0 (weight 8), etc
    - The salesman can keep re-visiting these cycles to get more $$:O



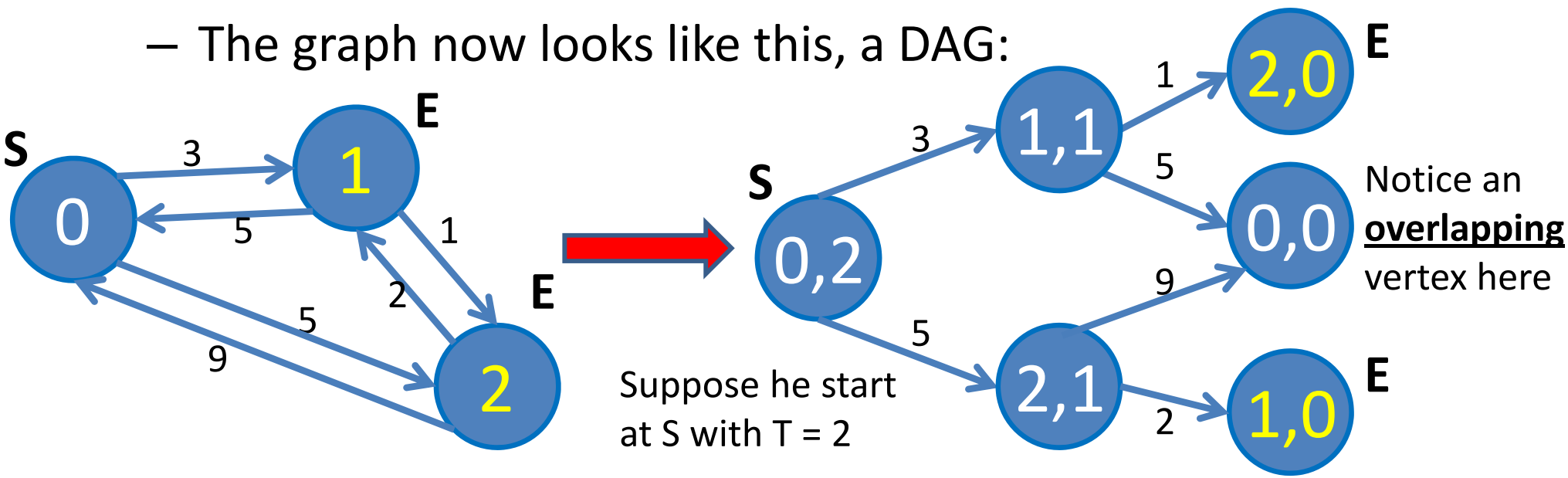
A Note About Longest Path Problem on General Graph

- The longest **(non simple)** path from S = 0 to any of the E will have $\propto$ weight
  - One can go through any **positive weight cycle** to obtain $\propto$
- 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 and requires "backtracking", as shown later
  - And we cannot write a recurrence if the graph is cyclic



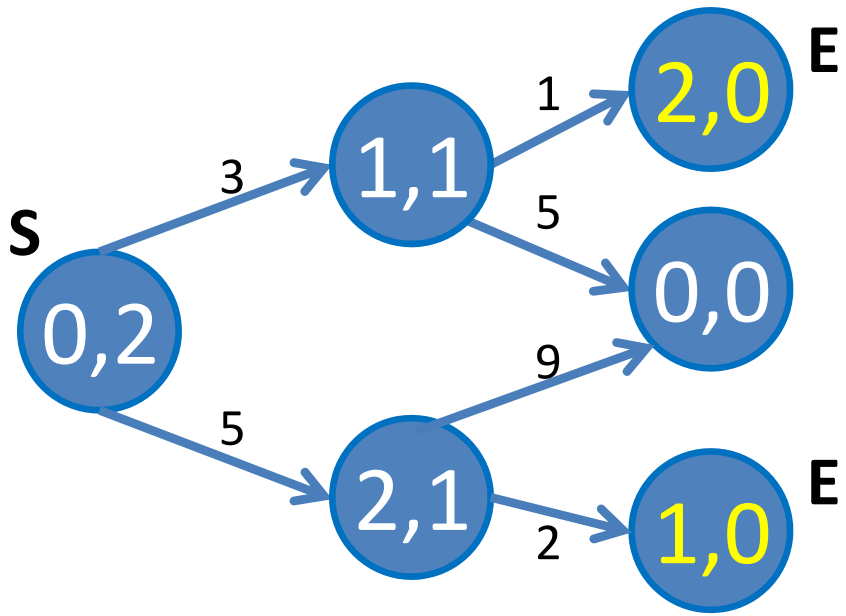
Conversion to a DAG

- The actual problem (UVa 10702) has an extra parameter that converts the general graph into 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
  - **KEY STEP**: Now each vertex has an 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 usually have to use the term SSL(simple)P on general graph



Graph Solution versus DP Solution

- What is the solution for the SSLP on DAG problem?
  - We are already familiar with this from previous lecture
  - SS Longest paths on DAG can be solved with either:
    - Find topological order and “stretch” edges according to this order/bottom-up DP (or sometimes called as ‘graph way’)
    - Or write a recursive function (top-down DP) with memoization
- But this problem is *harder* to be solved with graph way
  - The vertices now contain pair of information:
    - Vertex number and number of steps left
  - The number of vertices is not C, but now C*T
  - Pure graph implementation is going to be more difficult…

DP Solution (1)

- Let's solve this problem with top-down DP (recursion + memo)
  - We will *not* actually build the (implicit) DAG
- 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], we can ignore case where v = u

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]; // use a 2D array as memo

  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]; // to avoid re-computation
}
```

DP Analysis

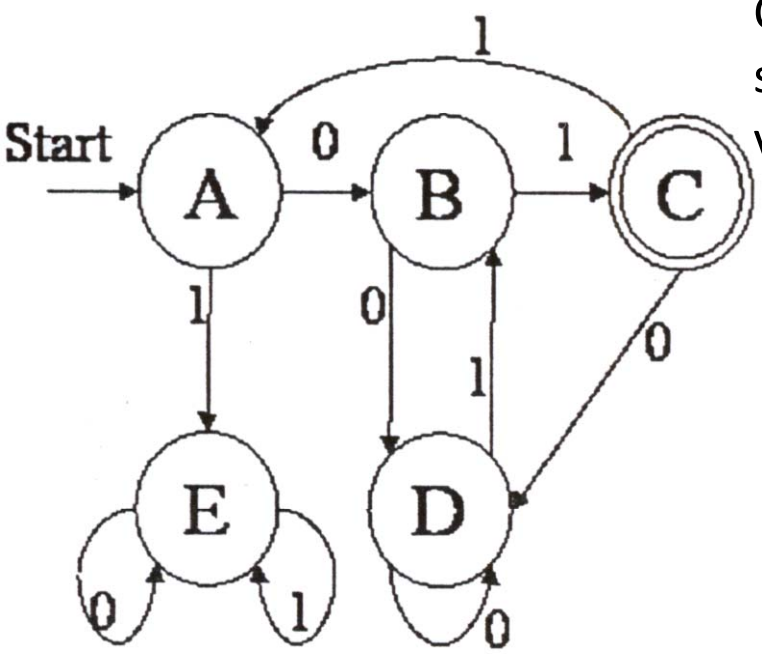
- What is the num of distinct states/space complexity?
  - That is, the vertices in the DAG
    - Answer: O(C*T)
- What is the time to compute one distinct state?
  - That is, the out-degree of a vertex
    - Answer: O(C), actually C - 1,but it is O(C)
- What is the overall time complexity?
  - That 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)

UVa 910 – TV Game

EXAMPLE 2

Another Problem

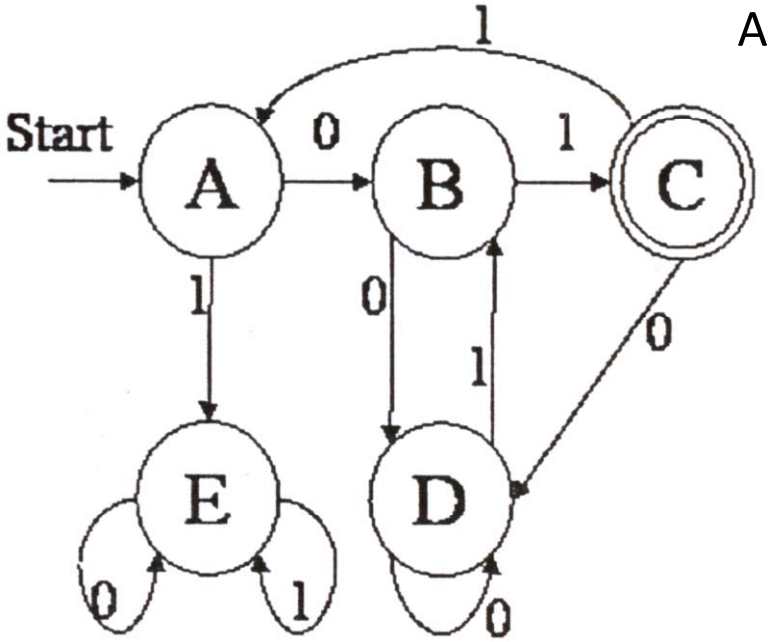
- UVa 910 – TV Game
  - There are **N** vertices (up to 26 vertices; labeled 'A' to 'Z')
    - Some of them are special (drawn with double circle)
  - Each vertex has exactly two outgoing edges
    - One that has label 0 and one that has label 1
    - The edge may be a self loop :O
      - Not a simple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



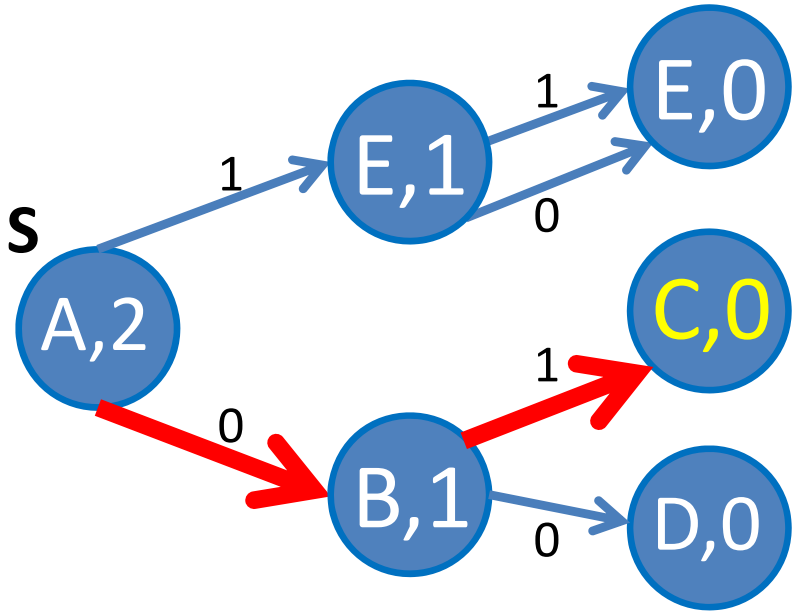
C is a special vertex

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 on General Multi-Graph
  - After conversion = Counting Paths on (non simple) 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 discuss solution with topological sort (graph way)
  - We will go straight to the solution with top-down DP (recursion)
- 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

Do you notice that “if0” and “if1” is another kind of Graph Data Structure?

DP Solution (2)

- In Java code (see UVa910.java):

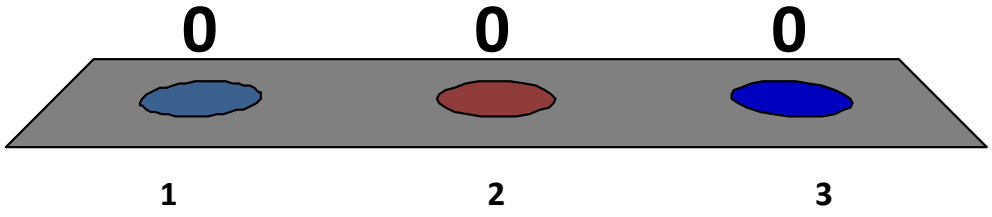
```
private static int ways(int u, int m_left) {
  if (m_left == 0) // no more move left?
    return special[u] ? 1 : 0;
  if (memo[u][m_left] != -1) // computed before?
    return memo[u][m_left];
  // two options, take edge 0 or edge 1
  // if0 and if1 is another "graph data structure"
  // store the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

So far…

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



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 ${}_{V}C_{2}$ pairwise distances
    - That is, a **complete weighted graph**
      - Which 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

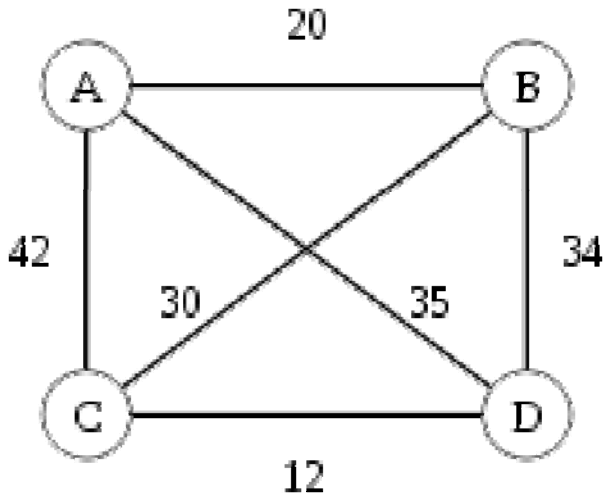
The shortest tour for this TSP instance is … (you will need some time to compute this)

1. Tour A-B-C-D-A with cost

   _____

2. Tour A-C-B-D-A with cost

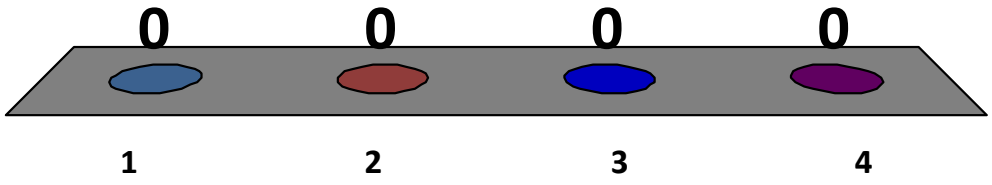
   _____



0 of 120

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

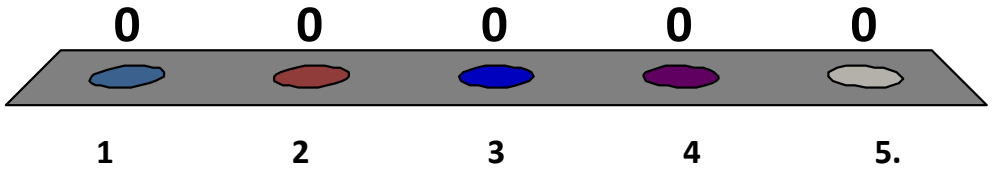
1. V valid tours
2. V^2 valid tours
3. V log V valid tours
4. V! valid tours



0 of 120

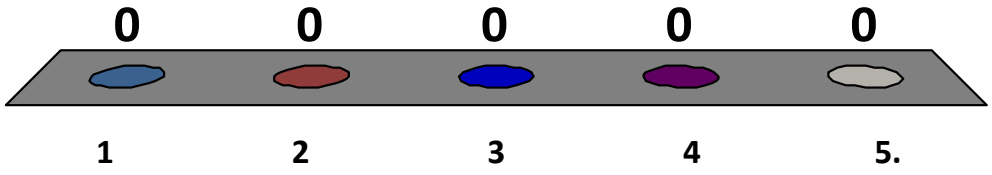
What is the value of “10!” ?

1. 10
2. 100
3. 3628800
4. 10000000
5. 9.332621544394415
   268169923885626 7
   e+157



What is the value of “100!” ?

1. 10
2. 100
3. 3628800
4. 10000000
5. 9.332621544394415
   268169923885626 7
   e+157



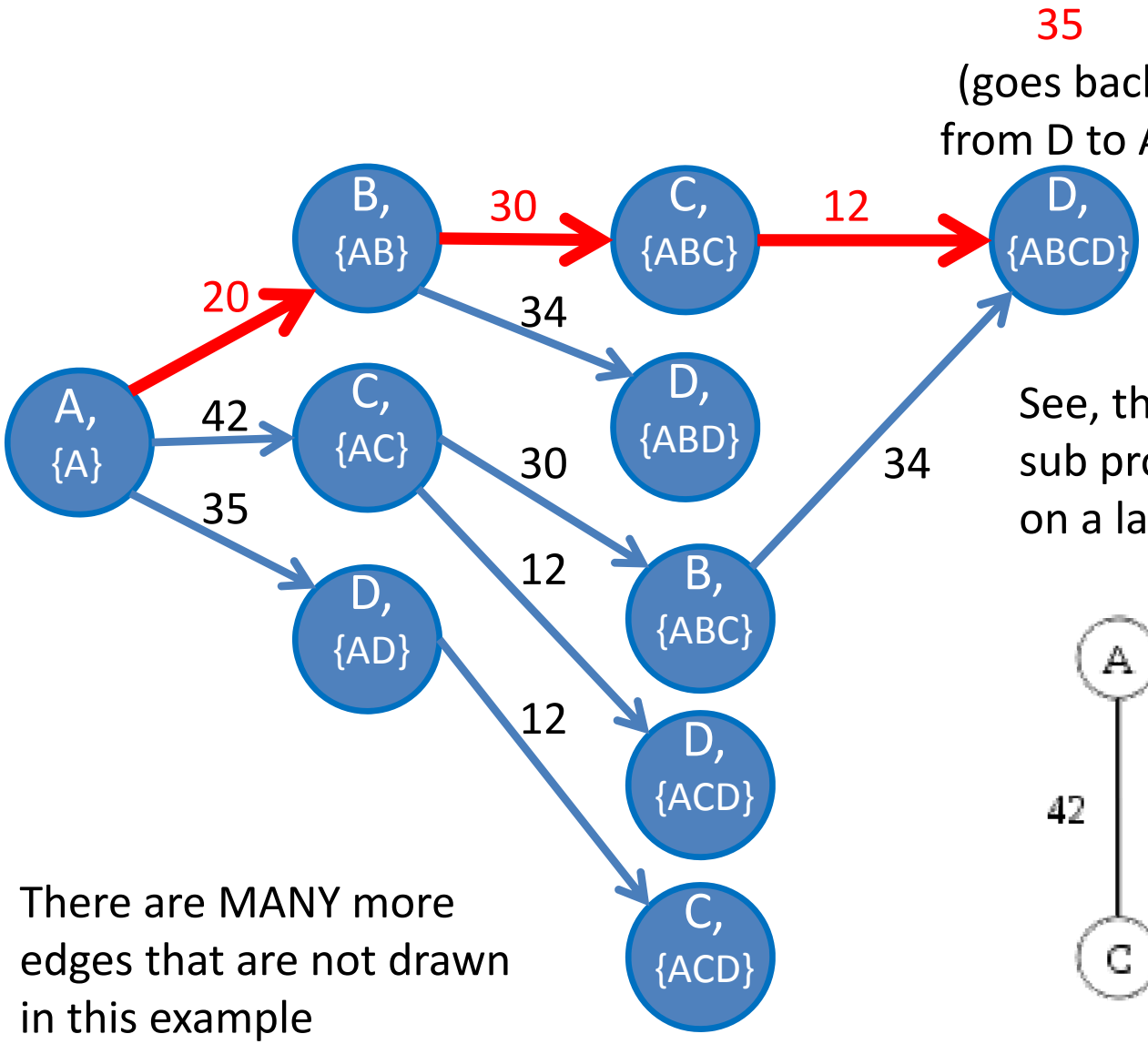
Brute Force (Naïve) Solution

- In sketch:
  - Try all V! tour **permutations**
  - Pick the one with the minimum cost…
- This sketch is too coarse for proper implementation
  - Perhaps writing a code that finds permutations of n objects is a bit hard to understand for first timers
    - (see TSPslow.java for a sample code)
    - I start from DFSrec (Lecture 05), O(V + E), V = N, E = N^2 ~ O(V^2)
    - I will show how to change DFSrec into a backtracking routine that tries all permutations, while still using the **visited** flag
    - This is an O(V!) algorithm, as there are V! possible tours

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)**
  - Backtracking is one example of Complete Search technique
- 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

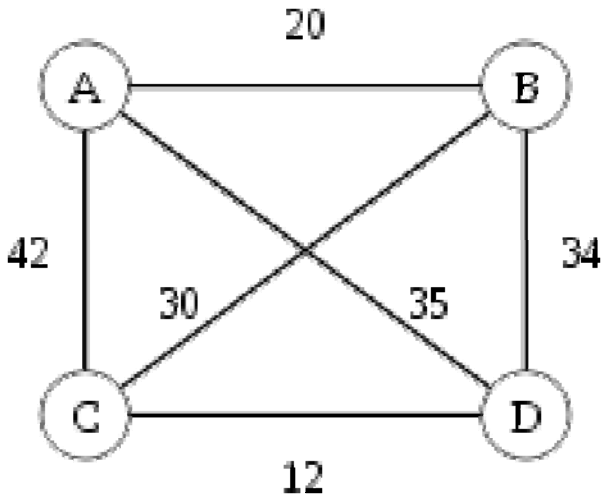
Converting to a DAG (2)



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

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 drawing a TSP DAG on a larger instance, say n = 10…



DP Solution (1)

- Now, how many vertices are there in this DAG?
  - $N * 2^N$
    - N is the current vertex
    - 2^N is the size of the set of visited vertices
- Then, how to store this "set of Boolean" effectively?
  - Introducing subset technique: **bitmask**
  - New "data structure" for lightweight set of Boolean

http://www.comp.nus.edu.sg/~stevenha/visualization/bitmask.html

New DS: lightweight set of Boolean

- An integer is stored in binary in computer memory
  - int x = 7_{10} (decimal) is actually 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
  - Theoretically up to 32 objects for a 32-bit integer
    - But we will only use up to 16 objects for TSP
  - 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

    - The result is 0_{10} = (00000_2), 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
    - The result is 8_{10} = (01000_2), that means bit i = 3 (from right) is **on**

Bit Operations (3)

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

    11001

    00100

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

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

Bit Operations (4)

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

    11001

    01000

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

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

Bit Operations (5)

- To turn on all bits of a set of Boolean with size **n**
  - **x = (1 << n) - 1**
  - Example:
    - n = 4
    - 1 << 4 = 10000
    - (1 << 4) - 1 = 1111
    - x = 1111, all n = 4 bits are turned on

http://www.comp.nus.edu.sg/~stevenha/visualization/bitmask.html

DP Solution (2) – in Pseudo Code

```
// memo is a 2D table of size N * 2^N

int DP_TSP(u, vis) // vis = visited
  if 'vis' is all 1 (in binary) // all vertices have been visited
    return weight(u, 0) // no choice, return to vertex 0
  if memo[u][vis] does not equals to -1 // computed before?
    return memo[u][vis]

  memo[u][vis] = INF // general case
  for each v in V // Q: do we need to check if (u, v) exists?
    if edge(u, v) exists and v is not turned on in bitmask 'vis'
      memo[u][vis] = min(memo[u][vis],
           weight(u, v) + DP_TSP(v, vis with v-th bit turned on);
  return memo[u][vis]

// note: You will implement this (and something else) in PS7/R
```

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-1
- What is the overall time complexity?
  - Answer: $O((N*2^N) * N) = O(N^2*2^N)$

Summary

- By definition, a general graph has cycle(s)
  - We cannot write a recursive formula in a cyclic structure…
  - Therefore we cannot use DP technique on general graph :O…
- In this lecture, we have seen how to convert *some* general graphs into DAGs by introducing (one) extra parameter
  - Now we can write recursive formulas and use DP technique
  - We can analyze the space and time complexity of DP solution easily
  - Some of these problems are better solved as top-down DP (written as recursive functions), i.e. we do not explicitly build the DAG…
  - Note that harder DP problems may require us to introduce more than one extra parameters… (not examinable for 100 marks for CS2010 tests, however optional bonus marks may be given for such subtask :D)