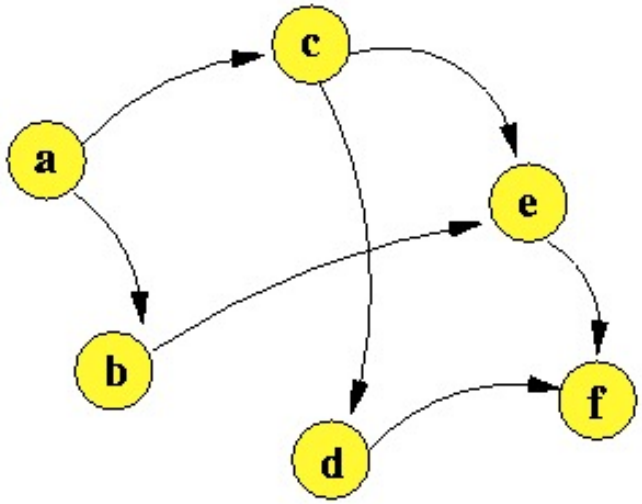
COMP20003 Workshop Week 11

1	Floyd-Warshall Algorithm & related topics
2	Graph Search & Gaming Assignment 3: Deadline= Friday 22nd October 2021 @ 5 pm (end of Week 12)
LAB	Lab: implement Floyd_Warshall Algorithm / Ass3

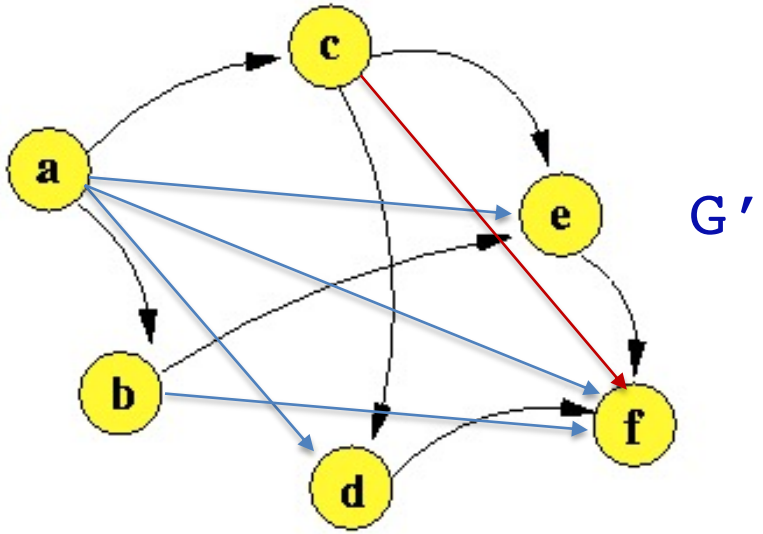
Transitive Closure of digraphs

Transitive Closure of a di-graph G:

- is a graph G′ where there is a link from i→j if there is a path from i→j in G.
- has an adjacency matrix A where $A_{ij}=1$ iif j is reachable from i in G.



G



G′

Related Tasks:

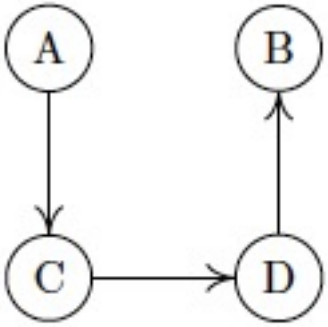
Compute the transitive closure for a digraph

Find APSP for a weighted graph

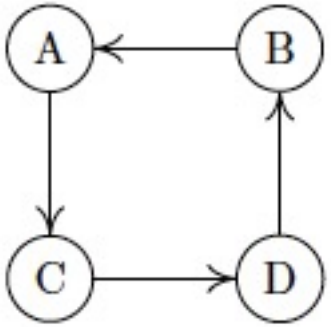
Examples: Transitive Closure of digraphs

Draw the transitive closure of the following two graphs:

(a)



(b)

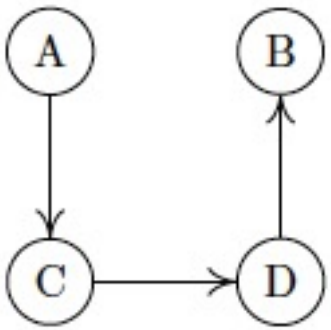


FROM	TO				
	A	B	C	D
A				
B				
C				
D				

Warshall's Algorithm: for Transitive Closure

Input: adjacent matrix A

Main argument: transitiveness: if there are paths i→k and k→j, then there is path i→j which uses k as an intermediate stepstone.



Warshall Algorithm

```
for (i=0; i<V; i++)
  // using i as intermediate
  for (s=0; s<V; s++)
    for (t=0; t<V; t++)
      if (A_si && A_it)
         A_st= 1;
```

Floyd-Warshall Algorithm

Purpose= ?

Similarity to Dijkstra = ?

Complexity = ?

Floyd-Warshall Algorithm - APSP

Given a weighted graph G=(V,E,w(E))

Find shortest path (path with min weight) between all pairs of vertices.

Recall: Dijkstra's Alg is SSSP – shortest path from a single source.

Idea:

- Can we use Dijkstra's Algorithm?
- Can we extend Warshall's Algorithm for this task?

Floyd-Warshall Algorithm

Find shortest path between all pairs (s,t) of vertices. That means minimizing `dist(s,t)`.

INITIALISATION: `dist= adjacency matrix of G.` That is:

```
dist(s,t) = w(s,t) or ∞
dist(s,s) = 0 for all s
```

IDEA for loops:

What if we use a particular node `i` as an intermediate stepstone in finding path from s to t?

```
for each pair (s,t) do
   if  s->i->t is better than s->t
      update dist(s,t)
```

Floyd-Warshall Algorithm

Main algorithm:

```
for each vertex i do
    for each pair (s,t) do
        if dist(s,i)+dist(i,t) < dist(s,t):
            update dist(s,t)
```

Conditions= ?

Data structures / Graph representation = ?

Complexity =

Floyd-Warshall Algorithm

Main algorithm:

```
initialize: dist= adjacency matrix
for each vertex i do
    for each pair (s,t) do
        if dist(s,i)+dist(i,t) < dist(s,t):
            update dist(s,t)
```

Conditions= ?

- directed or undirected
- weighted
- negative weights possible, but not in a cycle

Data structures / Graph representation = adjacency matrix

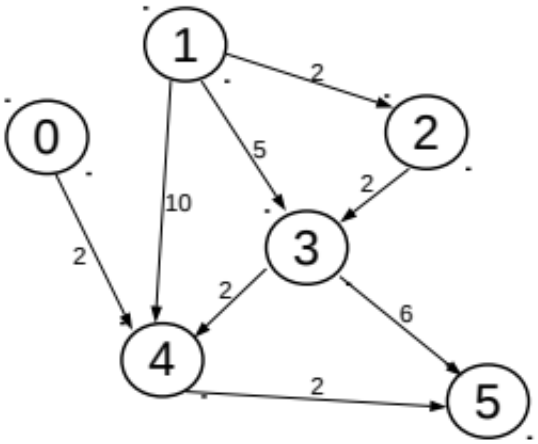
Complexity = $O(V^3)$

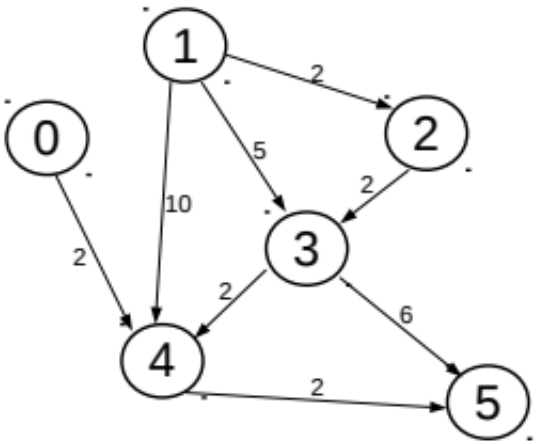
Floyd-Warshall Algorithm: write C code

How to retrieve path from s→t ?

```
initialize: dist= adjacency matrix
for each vertex i do
    for each pair (s,t) do
        if dist(s,i)+dist(i,t) < dist(s,t):
            update dist(s,t)
```

Run FW alg for



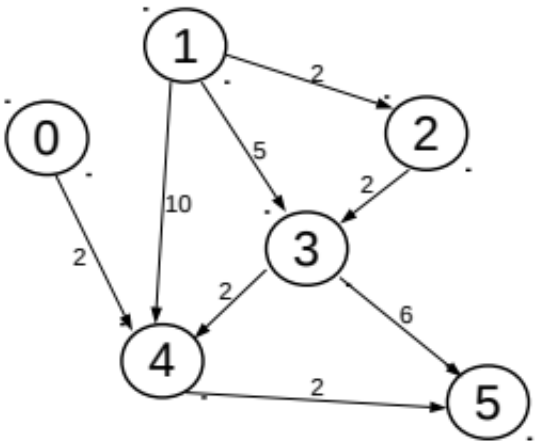


Draw the matrix representation. Trace the Floyd-Warshall algorithm.

TO

FROM

	0	1	2	3	4	5
0						
1						
2						
3						
4						
5						

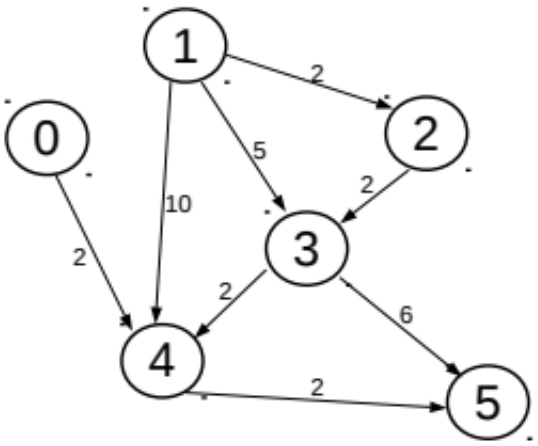


Trace the Floyd-Warshall algorithm.
Step i= 0, 1, 2, 3, 4, 5
TO

FROM

	0	1	2	3	4	5
0					2	
1			2	5	10	
2				2		
3					2	6
4						2
5						

empty cell for ∞
(note A[i][i] could be zero if we want)



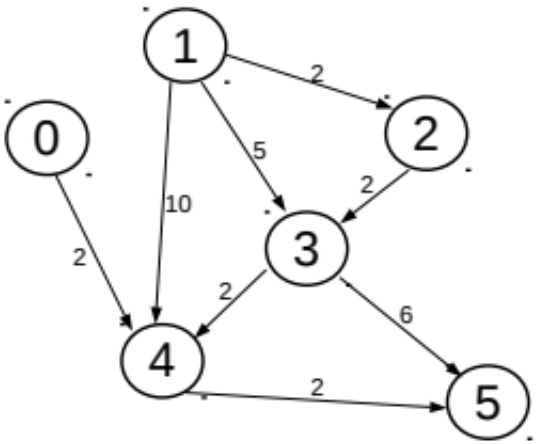
Trace the Floyd-Warshall algorithm (empty means ∞).
Step i= 0, 1, 2, 3, 4, 5

TO

FROM

	0	1	2	3	4	5
0	0				2	
1		0	2	5	10	
2			0	2		
3				0	2	6
4					0	2
5						0

Run FWA *manually*



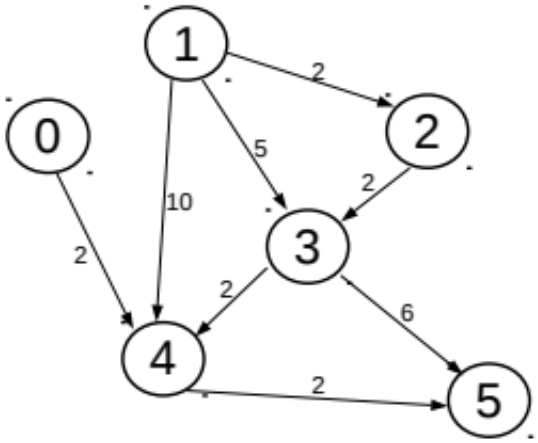
Notes:

- when 0, 1, or 5 is used as an intermediate, no change is possible (why?)
- using adj matrix, how do we recognize that 0, 1, 5 are not suitable?

	0	1	2	3	4	5
0	0				2	
1		0	2	5	10	
2			0	2		
3				0	2	6
4					0	2
5						0

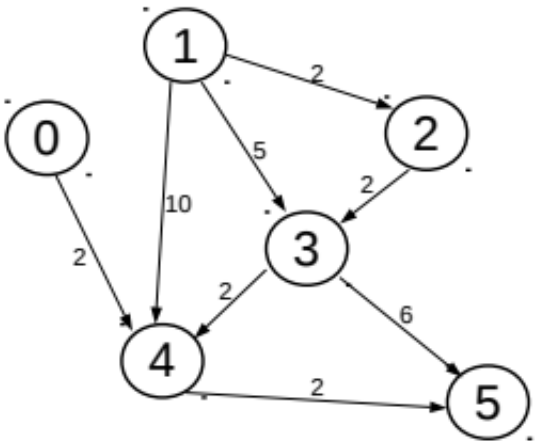
Run FWA *manually*

i= 2 as the stepstone



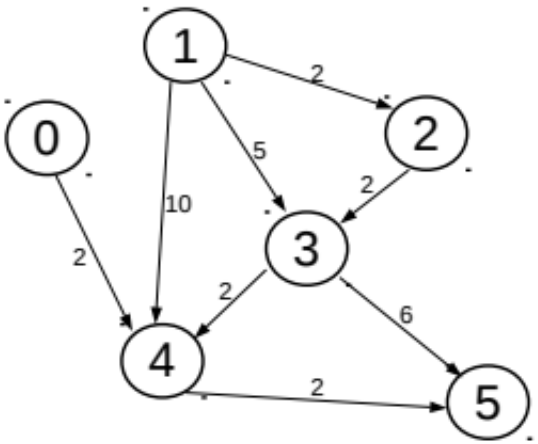
Only this cell need to be considered. Why?

	0	1	2	3	4	5
0	0				2	
1		0	2		10	
2			0	2		
3				0	2	6
4					0	2
5						0



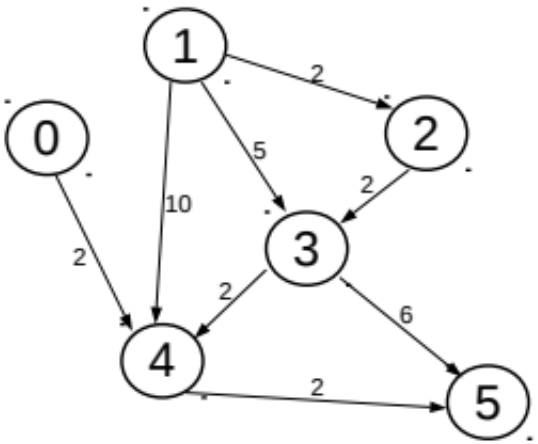
i= 2 as the stepstone

	0	1	2	3	4	5
0	0				2	
1		0	2	4	10	
2			0	2		
3				0	2	6
4					0	2
5						0



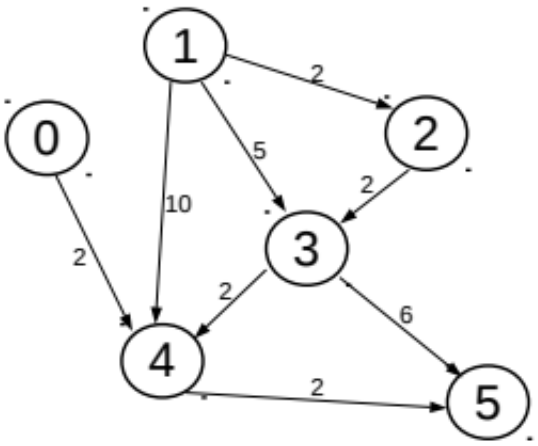
i= 3 as the stepstone

	0	1	2	3	4	5
0	0				2	
1		0	2	4	10	
2			0	2		
3				0	2	6
4					0	2
5						0



i= 4 as the stepstone

	0	1	2	3	4	5
0	0				2	
1		0	2	4	6	10
2			0	2	4	8
3				0	2	6
4					0	2
5						0



i= 4 as the stepstone

	0	1	2	3	4	5
0	0				2	4
1		0	2	4	6	8
2			0	2	4	6
3				0	2	4
4					0	2
5						0

choice of APSP algorithms for dence/sparse graph

FWA operates on adjacency matrix ad has complexity of $\boldsymbol{\theta}(V^3)$.

How about sparse graphs represented as an adjacency list? How would you approach the all pairs shortest paths problem?

APSP – comparing Dijkstra & Floyd-Warshall

When to apply?

- **DA:**
  - Type of graphs:
  - Specific requirements:
- **FW:**
  - Type of graphs:
  - Specific requirements:

Big-O complexity

(supsing to apply Dijkstra for the APSP task)

	Dijkstra	Floyd-Warshall
General		
Sparse		
Dense		

Q11.1: APSP – comparing Dijkstra & Floyd-Warshall

Big-O complexity
(supposing to apply Dijkstra for the **APSP task**)

	Dijkstra	Floyd-Warshall
General	$V\,(V+E)\log V$	V^3
Sparse	$V^2 \log V$	V^3
Dense	$V^3 \log V$	V^3

Graph Search: some interesting tasks

- Find a longest path in a weighted (acyclic) graph
- Find an Euler cycle
- Find a Hamiltonian cycle
- …

An **Euler trail** is a way to pass through every edge exactly once. If it ends at the initial vertex then it is an *Euler cycle*. Note that an Euler trail might pass through a vertex more than once.

A **Hamiltonian path** is a path that passes through every vertex exactly once (NOT every edge). If it ends at the initial vertex then it is a *Hamiltonian cycle*. Note that a Hamiltonian path may not pass through all edges.

Q: Why an Euler trail, but not a Hamiltonian path, must pass through every edges exactly once?

A: Because Euler and Edge share the same starting letter E 

P, NP, NP: Currently Naïve & Intuitive Understanding

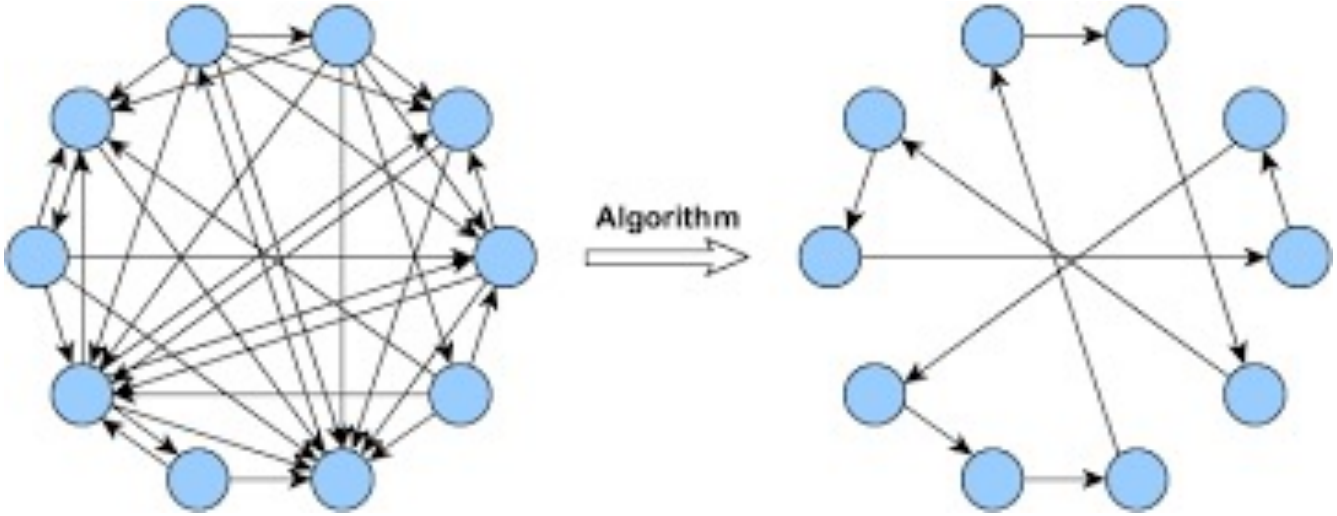
Decision Problem: Problem with YES/NO answer. A decision problem is:

- P: iif it can be **solved** in *polynomial time* by an algorithm.
- NP: iif the 'yes'-answers can be ***verified*** in polynomial time ($O(n^k)$ where n is the problem size, and k is a constant).
- NP-Complete: iif it's NP and, (so far) there is no polynomial time algorithm for solving.

Notes:

- The above concepts are for intuitive understanding, and are not the correct definitions.
- For true definitions of P, NP and related interesting theoretical topics: attend live lecture W12.
- Grady's slides this week also addresses a little bit on these concepts

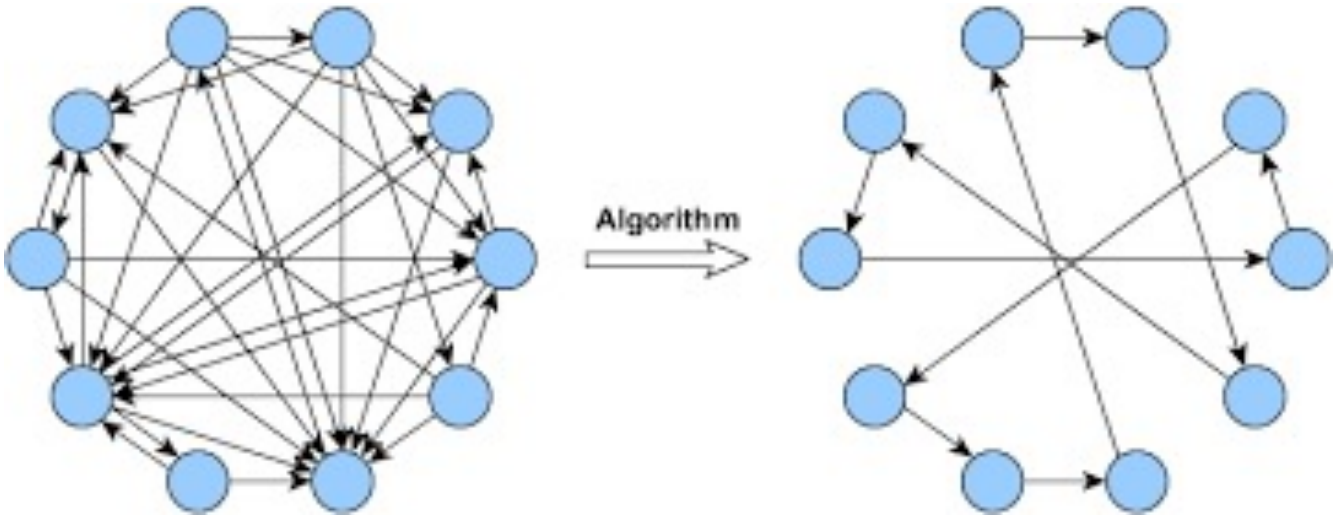
Graph Search can be a NP-complete task: Hamiltonian cycle



Can we just run DFS or BFS? The complexity would be O(V+E), right?

Picture from: https://archive.codeplex.com/?p=findinghamiltoniancycle

Graph Search can be a NP task: Hamiltonian cycle



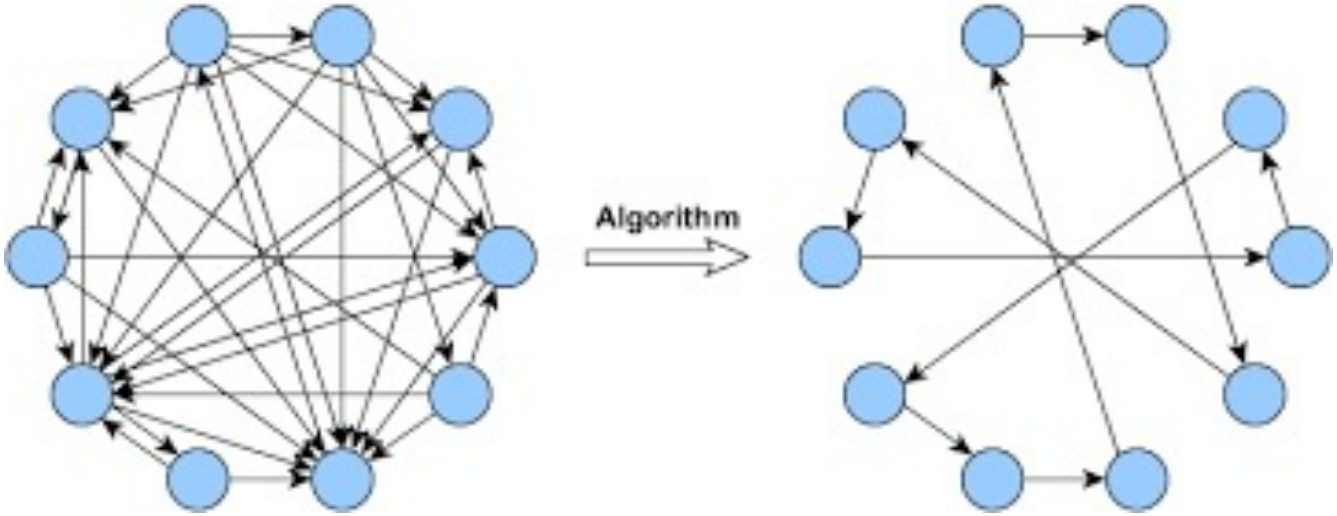
Yes, we can do the path finding in DFS or BFS manner, but the complexity is no longer O(V+E). *Why?*

The complexity of this task is O(?)

The task belongs to the NP-Complete class.

Picture from: https://archive.codeplex.com/?p=findinghamiltoniancycle

Graph Search can be a NP task: Hamiltonian cycle



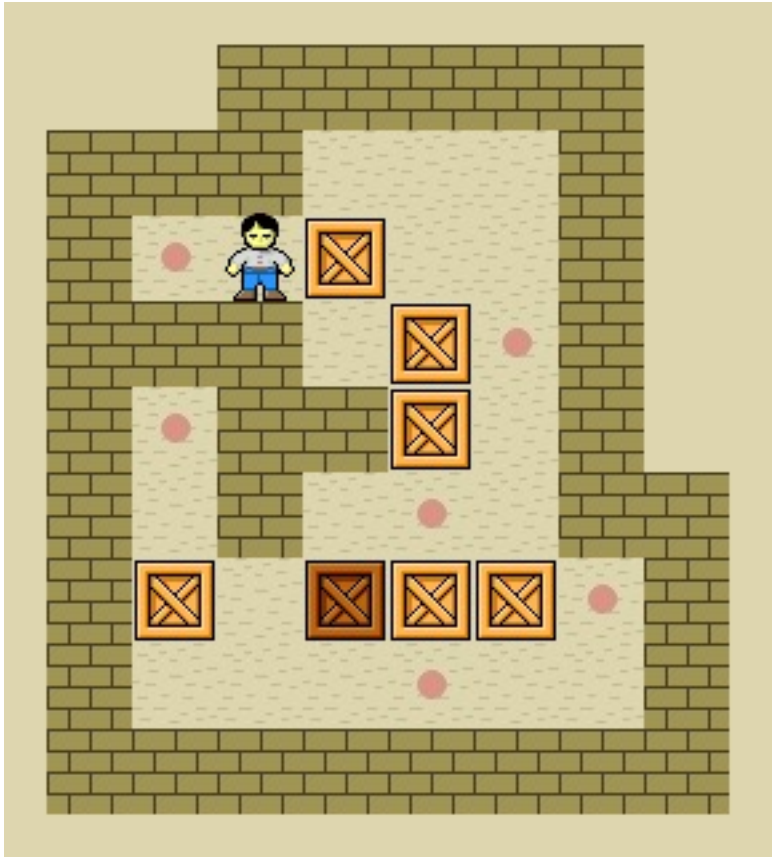
Yes, we can do the path finding in DFS or BFS manner, but the complexity is no longer O(V+E). *Why?*

The complexity of this task is O(?)

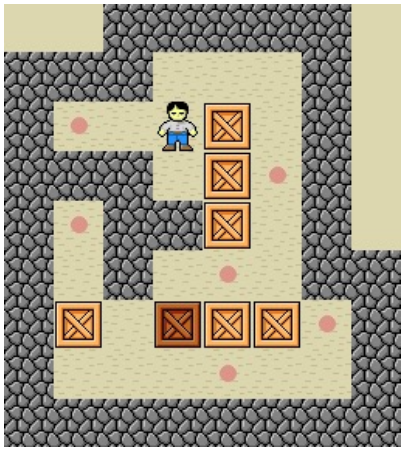
The task belongs to the NP-Complete class. *What's that?*

Picture from: https://archive.codeplex.com/?p=findinghamiltoniancycle

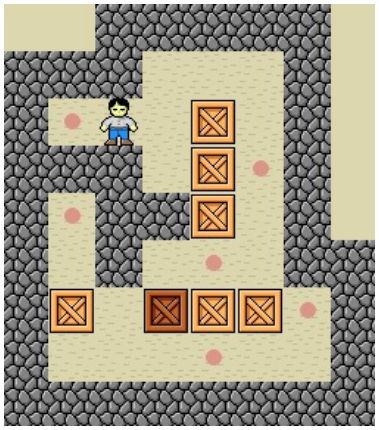
Games & Implicit Graphs: The Sokoban



Games & Implicit Graphs [using ass2]



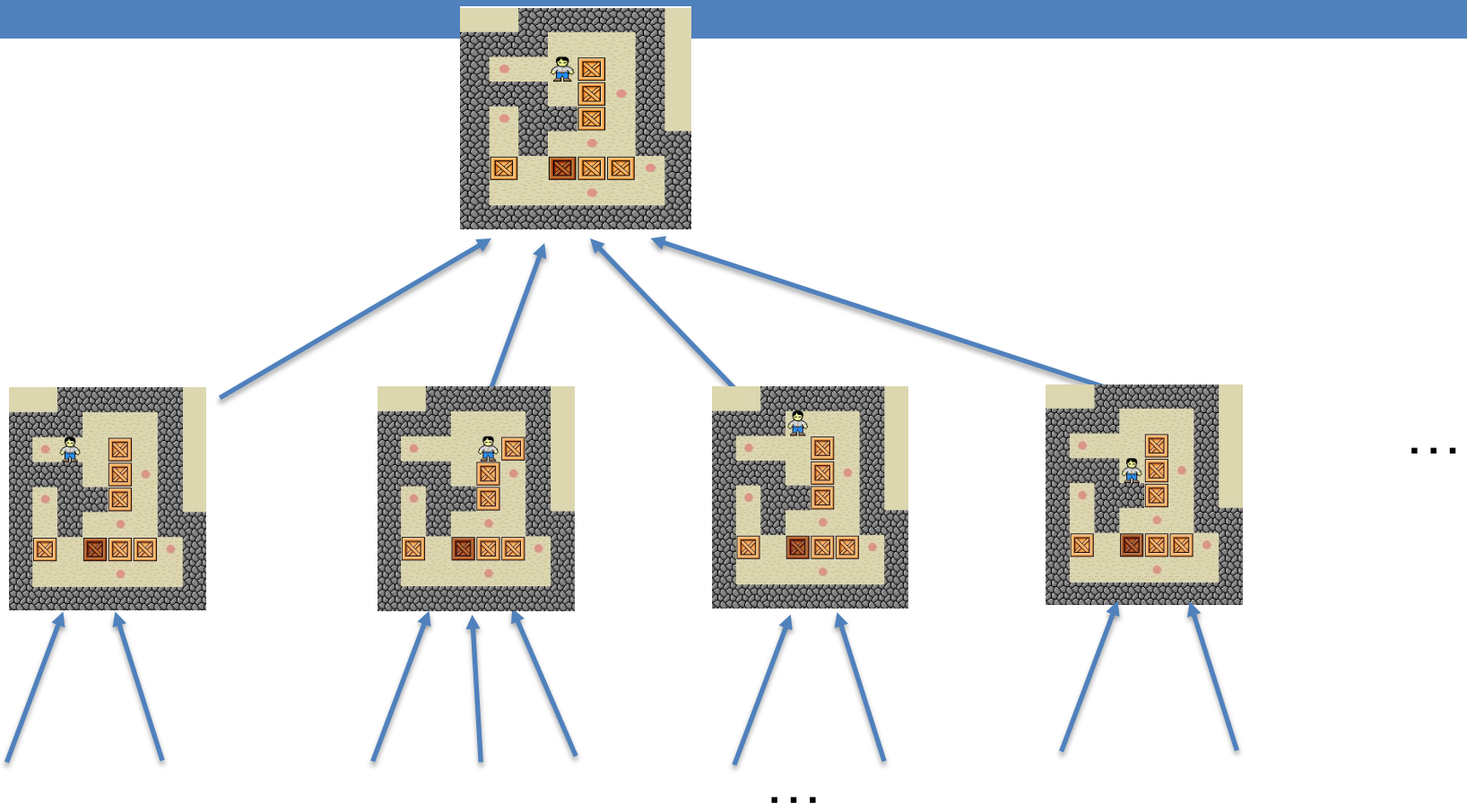
go left



We represent the game implicitly as a graph:

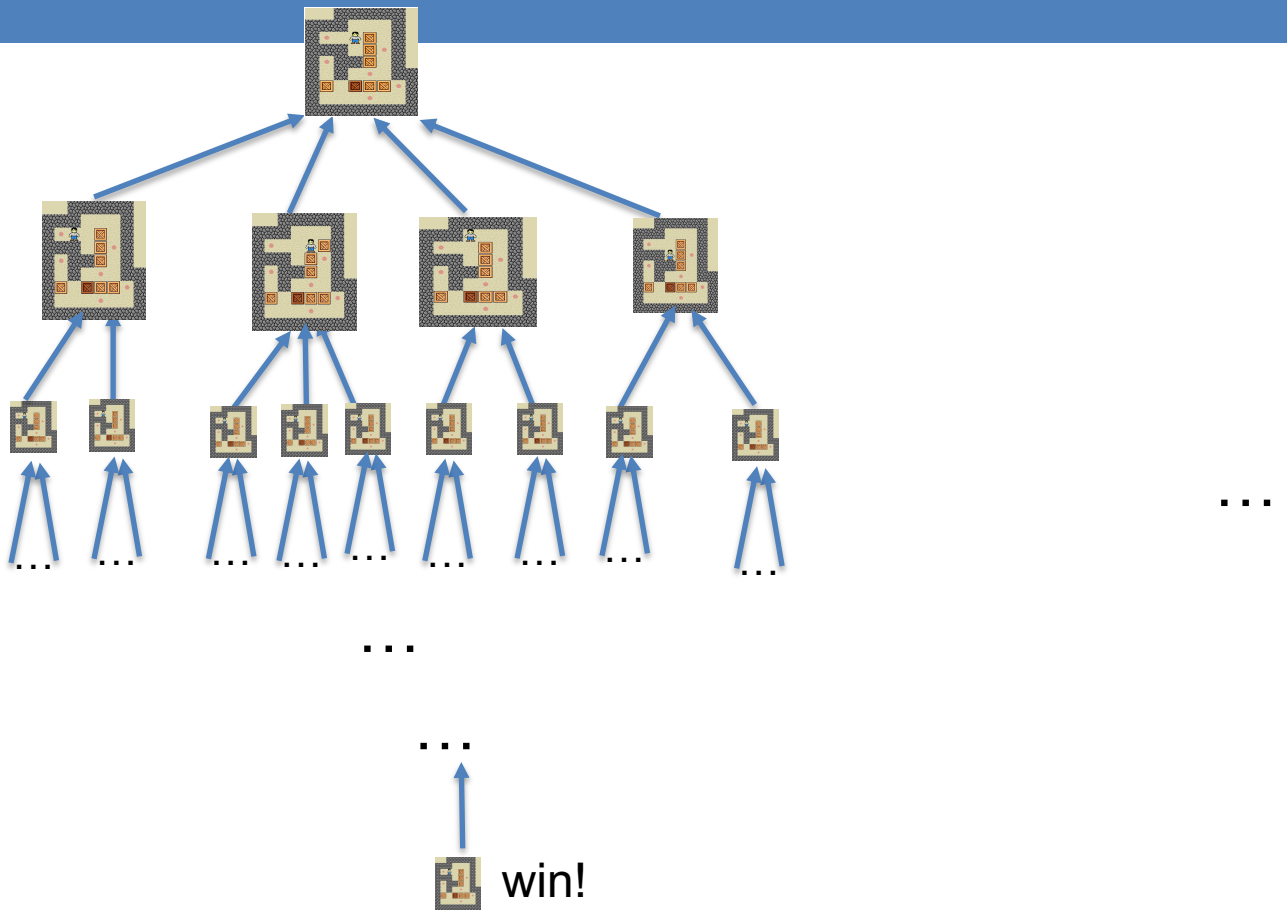
- A particular configuration of the game board is called a *state*
- When a move is performed, the board goes from one state to another state
- So: **A state is a vertex, and a move is an edge of the graph**

Note: For simplicity/convenience, in addition to the board configuration, some additional elements were added to each vertex.



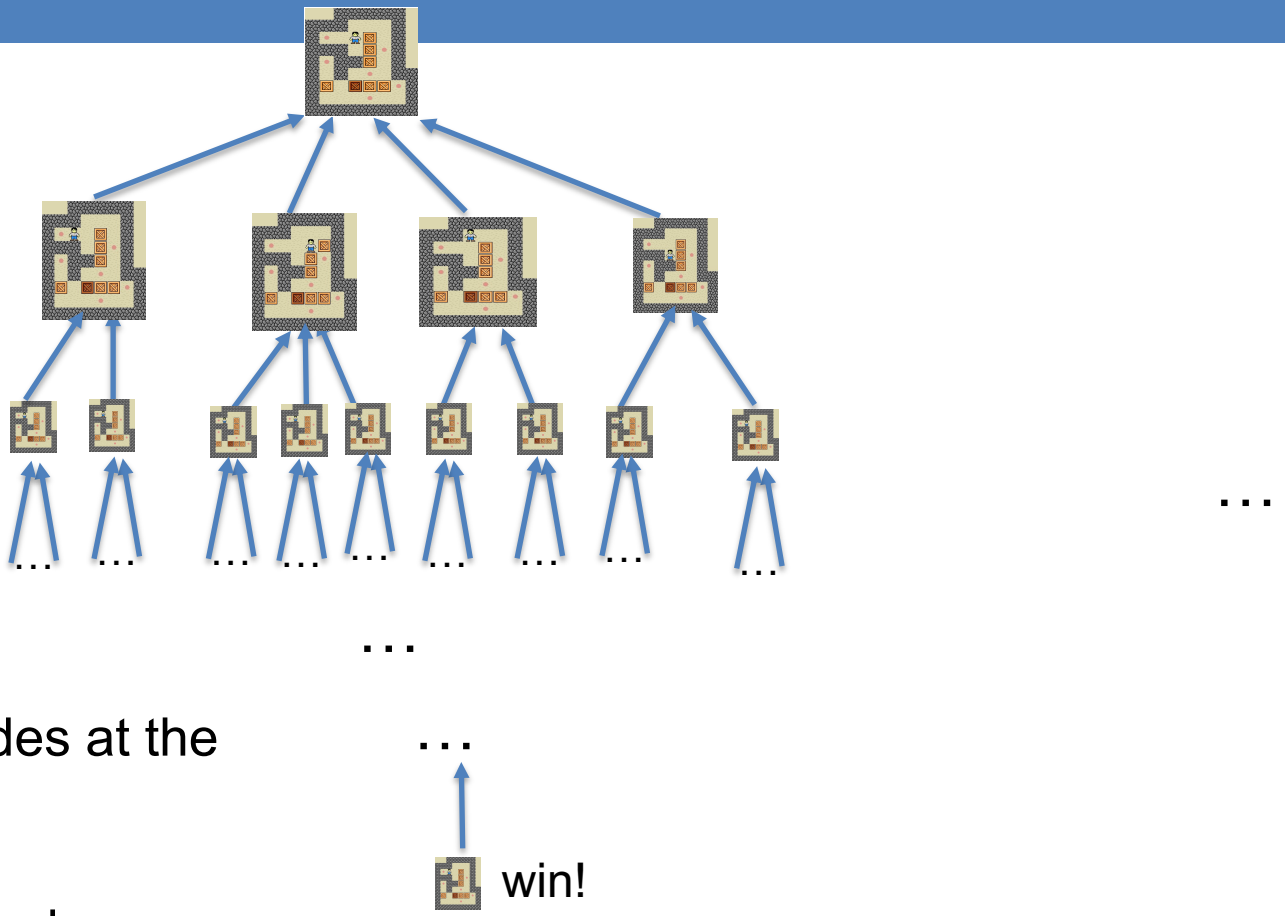
We look at the state and generate all possible actions from that state, these turn into edges

With implicit graphs we usually don't generate the whole graph



Q:

- What's the maximal depth of the search?
- number of nodes= ?
- Complexity=? P? NP-Complete?



How to free all the nodes at the end?

Keeping track of the nodes:

- After generating a valid, new node:
  - push it into the PQ
- After popping a node from a queue:
  - can we free it?
  - then what to do with it?

The Task: see ED

Additional Notes

- Attend online Week 12 lecture: Computational Complexity, P & NP
- Grady's this week workshop slides contain some more explanations on P, NP and graph search
- Check out https://clementmihailescu.github.io/Pathfinding-Visualizer/ - a pretty interactable visualization tool for all the traversals and searches.
- The above is a fork of the project:

  https://qiao.github.io/PathFinding.js/visual/