COMP20003 Workshop Week 11

1	Floyd-Warshall Algorithm
2	Graph Search
3	Assignment 3

Floyd-Warshall Algorithm

Purpose= ?

Similarity to Dijkstra = ?

Complexity = ?

Floyd-Warshall Algorithm - APSP

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

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

Idea= ?

Floyd-Warshall Algorithm

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

At the start `dist` is the adjacent matrix:

```
if s==t: dist(s,s) = 0
```
, otherwise

`dist(s,t) = w(s,t)` OR `dist(s,t) = ∞`

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: write C code

Supposing: A[][] is the adjacent matrix, V is number of vertices. Write C code for the algorithm:

```
for (i=0; i<V; i++) {
   // use i as stepstone
   for (s=0; s<V; s++) {
      for (t=0; t<V; t++) {
         if (dist[s][i] +  dist(i,t) < dist(s,t)) {
             dist[s][j]=…
             path[s][t]=…
         }
      }
}
```

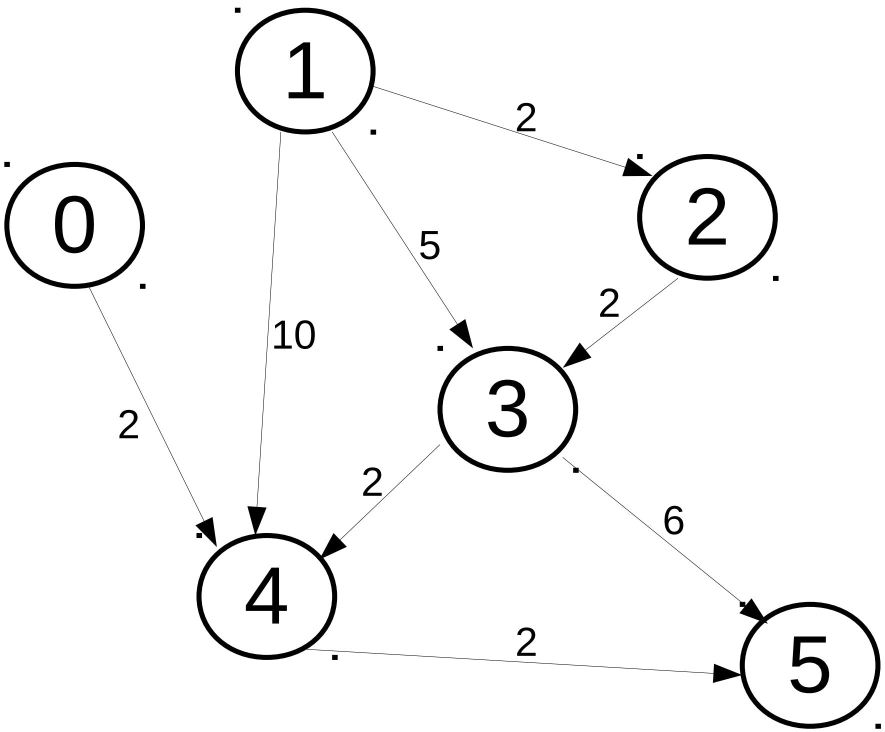
How to retrieve path from s→t ?

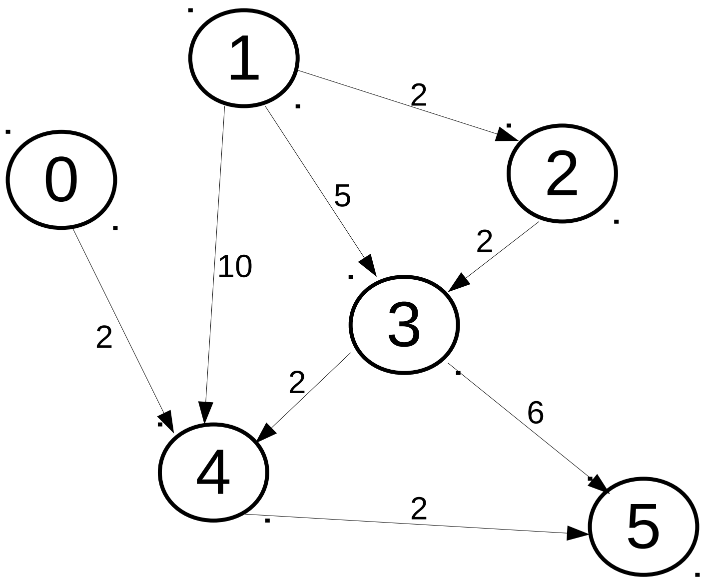
Floyd-Warshall Algorithm

Floyd-Warshall algorithm (weights, **`A[i][i]`**= 0, no path=∞)

```
for(i=0;i<V;i++)
  for(s=0;s<V;s++)
    for(t=0;t<V;t++)
       if(A[s][i]+A[i][t] < A[s][t])
            A[s][t] = (A[s][i]+A[i][t]);
```

Run FW alg for



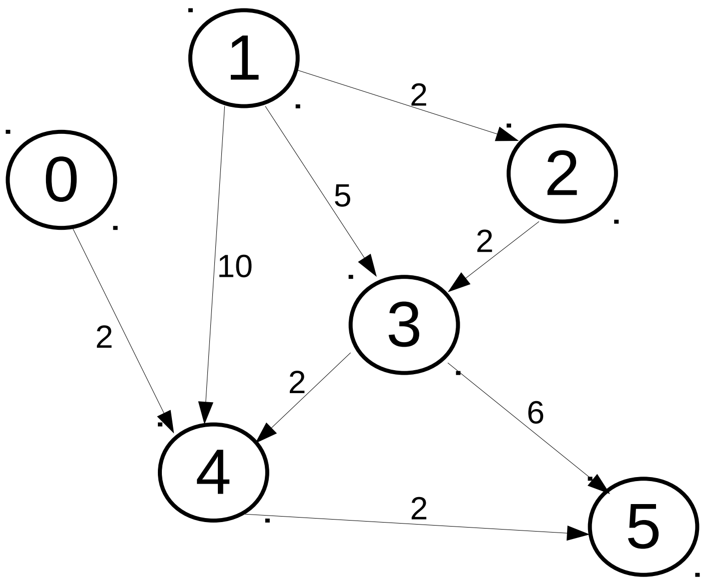


Draw the matrix representation.
Trace the Floyd-Warshall algorithm.

TO

FROM

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



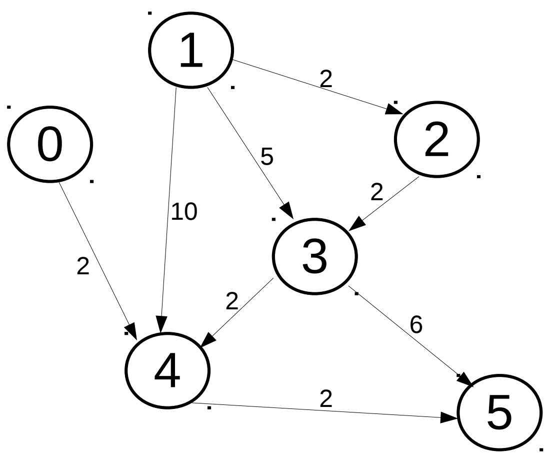
Trace the Floyd-Warshall algorithm.
Step i= 0, 1, 2, 3, 4, 5

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

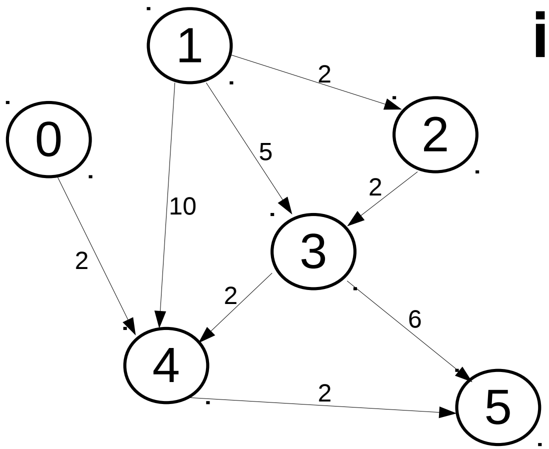
	0	1	2	3	4	5
0	0				2	
1		0	2	5	10	
2			0	2		
3				0	2	6
4					0	2
5						0

Run FWA *manually*

Note:
Nochange to the matrix when node 0, 1, or 5 is employed as an intermediate stepstone (why?)

	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*

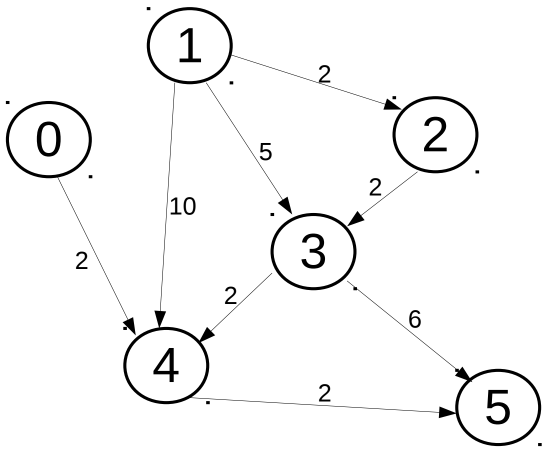


i= 2 as the stepstone

Only this cell need to be considered. Why?

`i= 2` as the stepstone: we use column `2` (== which nodes can lead to `i`) and row `2` (which nodes `i` can reach to) as references.

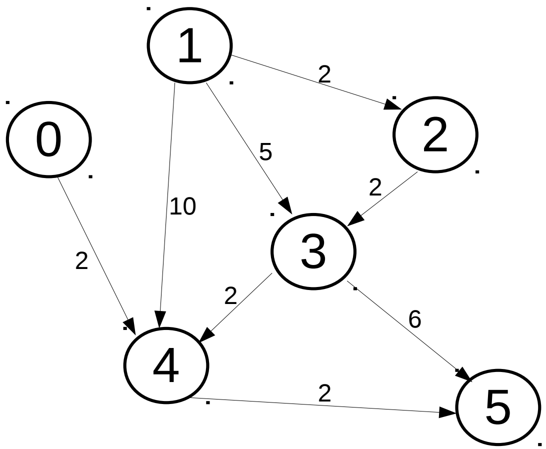
	0	1	2	3	4	5
0	0				2	
1		0	2	5	10	
2			0	1 2		
3				0	2	6
4					0	2
5						0



`i= 2` as the stepstone:

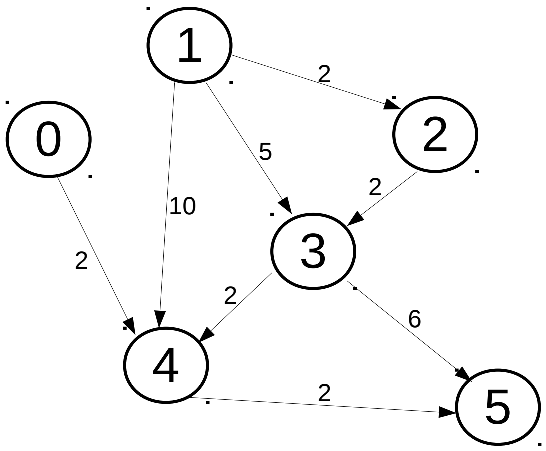
At that cell the referenced values are **2** and **2**, so we can have new cost **2+2**= **4**, which is better than the current value **5** → replace.

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

FWA: Complexity

choice of APSP algorithms for dence/sparse graph

FWA operates on adjacency matrix and 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?

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

Big-O complexity

(supposing to apply Dijkstra for the APSP task)

	Dijkstra	Floyd-Warshall
General	O(V (E+V)log V)	T(V^3
)Sparse	O(V(V+V) logV)= V^2logV	V^3
Dense	O(V(V^2+V) logV)= V^3logV	V^3

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

The Task: Path Finding

finding a path P from node S to node G, so that P satisfy some condition:

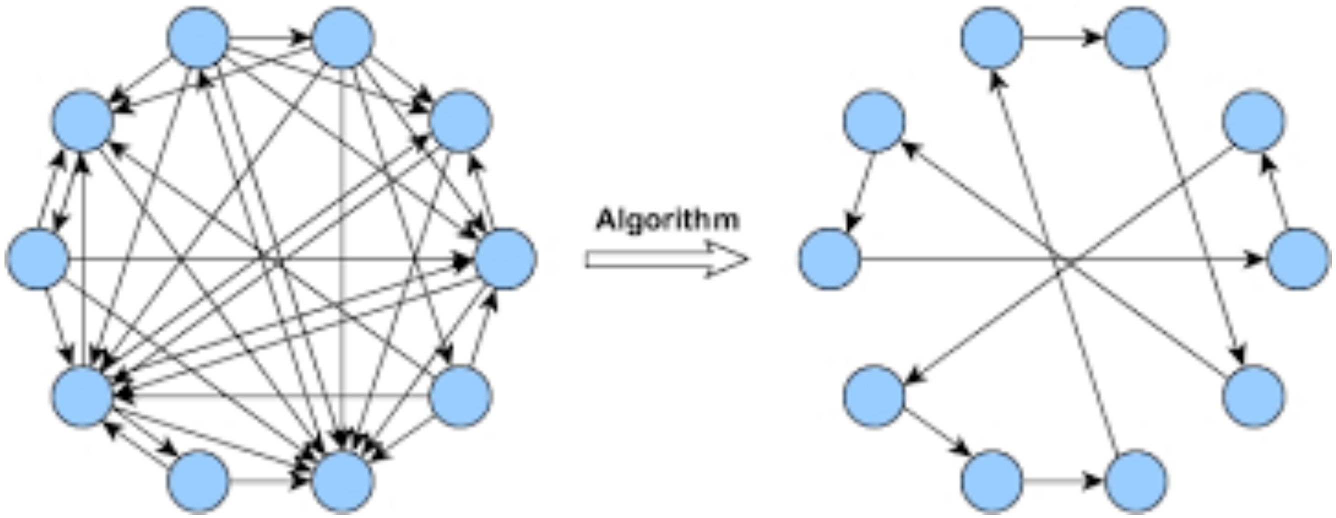
P is $S = s_0 \rightarrow s_1 \rightarrow s_2 \rightarrow \ldots \rightarrow s_k = G$

condition: for example minimize the sum of weights, minimize the max edge weight.

Example: weighted graph with vertices are cities, an edge represent the road between the 2 cities, and weight is just road length. We want to find a) the shortest part from MEL to SYD, b) the path that we surely stop at a city after 8 hours of driving.

Algorithm? We can change Dijkstra algorithm to serve this purpose.

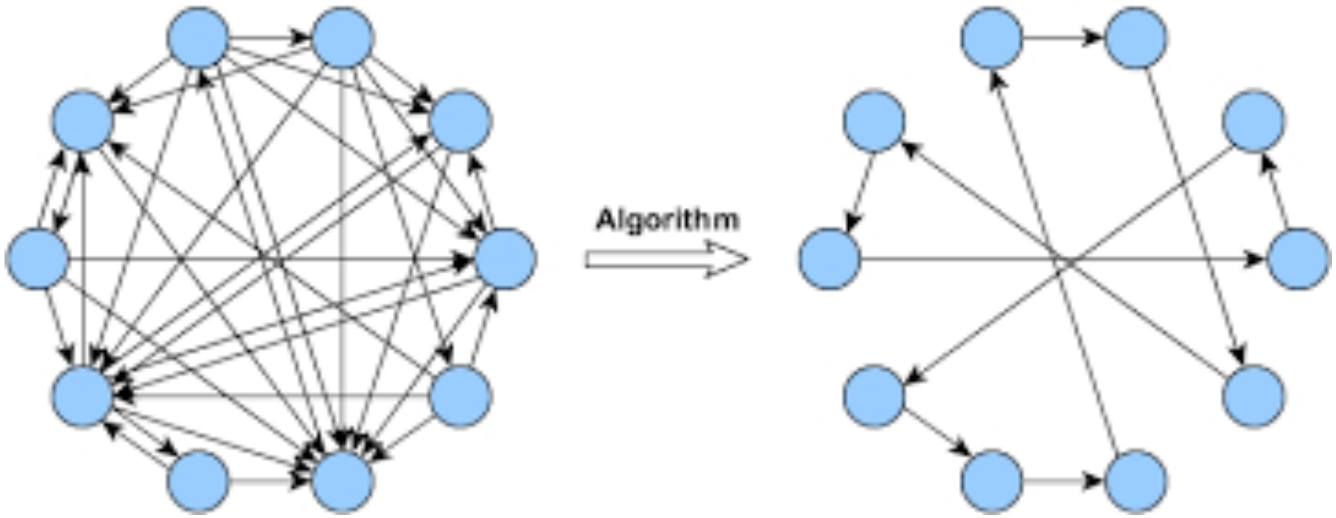
Graph Search can be a NP task: Hamiltonian cycle



Can we just run DFS? 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



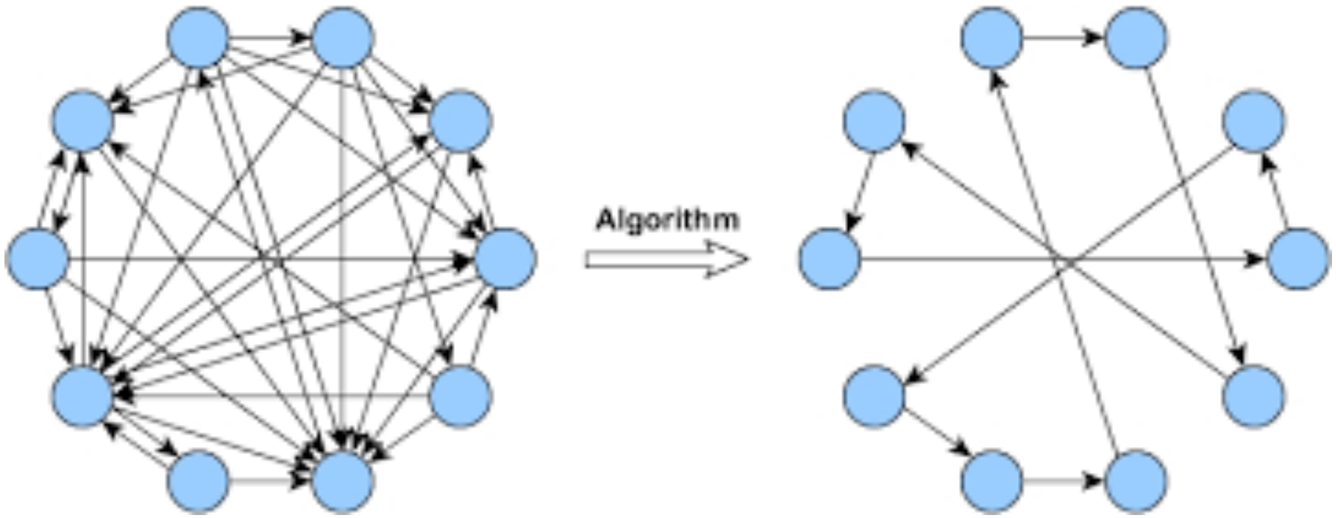
Yes, we can do the path finding in DFS or BFS manner, but the complexity is no longer O(V+E). *Why?*

The complexity of this task is O(?)

The task belongs to the NP-Complete class. P NP

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(?)

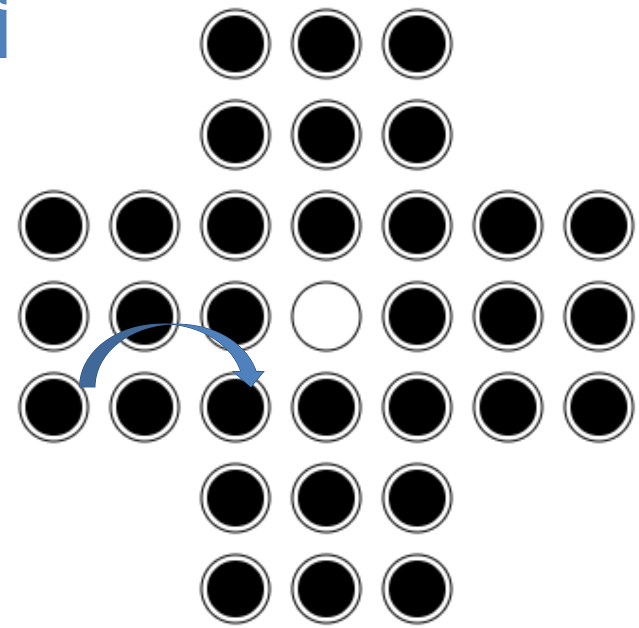
The task belongs to the NP-Complete class. *What's that?*

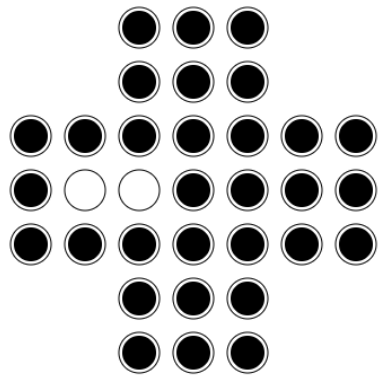
Picture from: https://archive.codeplex.com/?p=findinghamiltoniancycle

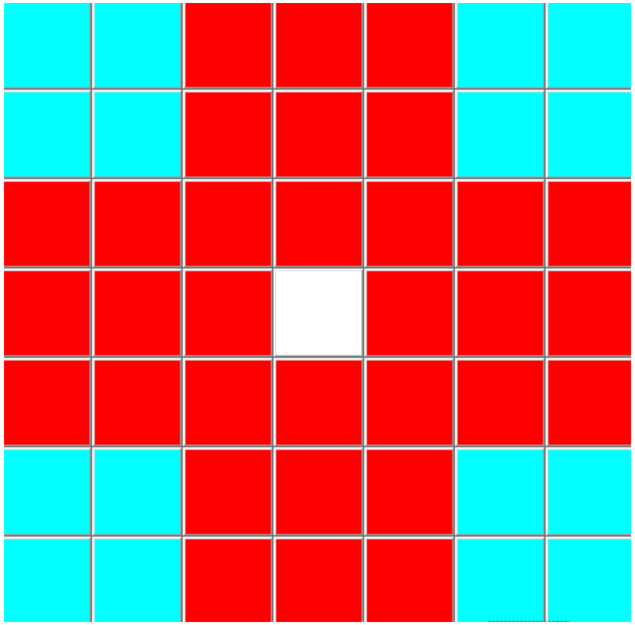
Gaming & Implicit Graphs [using ass2]

The Peg Solitaire game:
The player can move a peg jumping on top of another adjacent peg, if there is a free adjacent cell to land. There are 4 valid jumps: Left, Right, Up and Down.
The objective is to clean the board until there is only 1 peg left.

- How to represent (the process of running) the game as a graph?
- What does that mean “search for a solution” in this case?







Source: https://github.com/mzmousa/peg-solitaire

The Peg Solitaire Game as an implicit graph

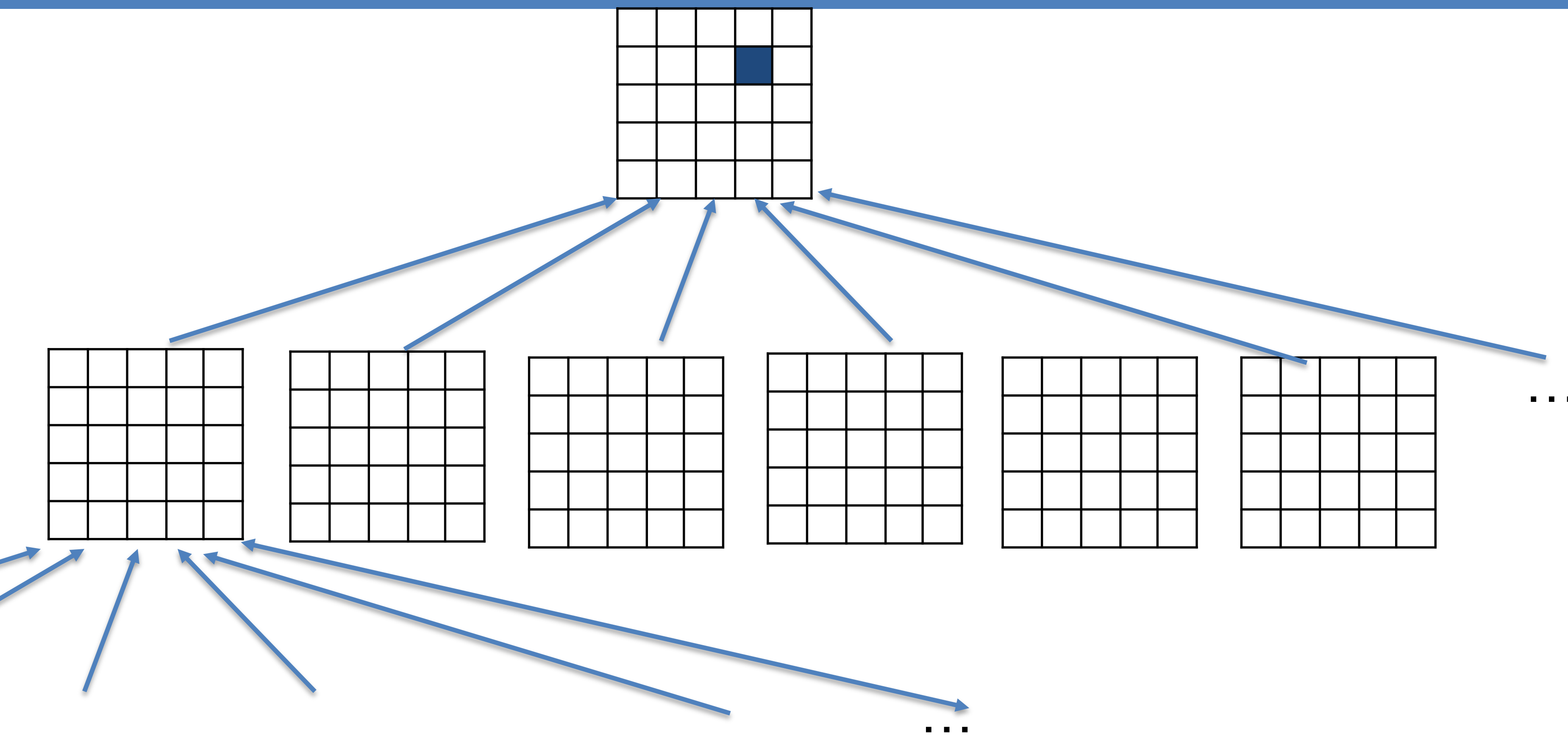
We represent the game implicitly as a graph:

- 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
- A state is a node, 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 state:

Each possible configuration of the Peg Solitaire (Pegsol) is a tuple made of: $m \times m$ grid board, the position of the cursor and whether the peg under the cursor has been selected.

The Pegsol Graph $G = \langle V,E \rangle$ is implicitly defined. The vertex set V is defined as all the possible configurations (states), and the edges E connecting two vertexes are defined by the legal jump actions (right, left, up, down).



Q:

- What happen after a legal move done?
- What's the maximal breath of the search?
- What's the maximal depth of the search?
- Complexity=? P? NP?

The Task

*Your task is **to find the path leading to the best solution**, i.e. leading to the vertex (state) with the least number of remaining pegs. A path is a sequence of actions. You are going to use Depth First Search to find the best solution, **up to a maximum budget** of expanded/explored nodes (nodes for which you've generated its children).*

*S*tart with the initial configuration (done for you)

When the AI solver is called (Algorithm 1), it should explore all possible paths (sequence of jump actions) following a Depth First Search (DFS) strategy, until consuming the budget or until a path solving the game is found.

Optimization: *Note that **we do not include duplicate states in the search**. If a state was already generated, we will not include it again in the stack (line 21).*

The algorithm should return the best solution found, the path leading to the least number of remaining pegs. This path will then be executed by the game engine.

The Task

You might have multiple paths leading to a solution. ***Your algorithm should consider the possible action by scanning the board in this order****: traverse coordinate x = 0,...,m first, and then y = 0, . . . , m looking for a peg that can jump, and then selecting jumping actions left, right, up or down.*

Make sure you manage the memory well: When you finish running the algorithm, you have to free all the nodes from the memory, otherwise you will have memory leaks. You will notice that the algorithm can run out of memory fairly fast after expanding a few milion nodes.

When you applyAction you have to create a new node, that

1. *points to the parent,*
2. *updates the state with the action chosen,*
3. *updates the depth of the node.*
4. *updates the action used to create the node*

We’re joining a team-work programming project, just like the software industry…

Then explore the package, know your work, and know what tools other members offer…