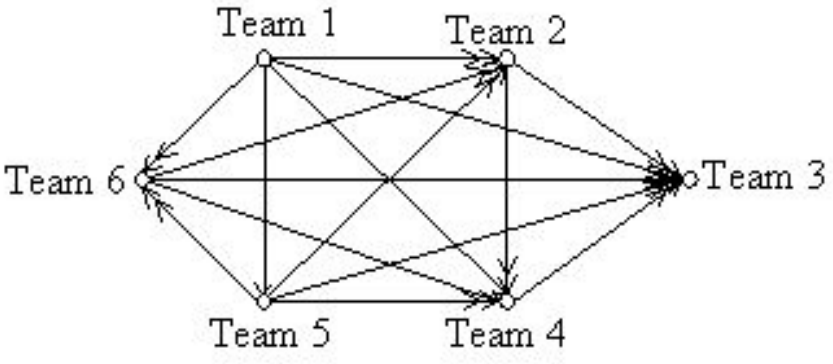
Graph Theory Term-Paper Presentation

Robin Babu P (17CS10045)
Kousshik Raj (17CS30022)

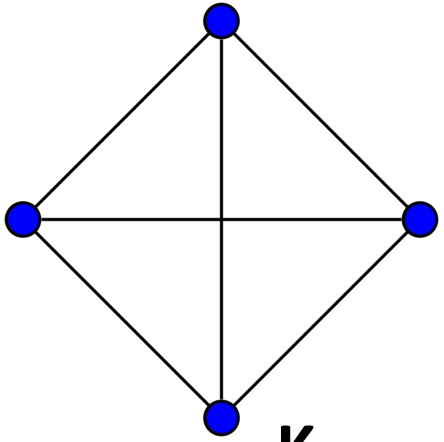
Problem

Implementation of Algorithms on Tournaments in Directed Graphs.

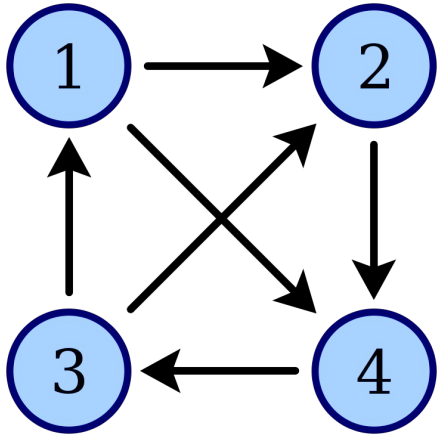


Tournament

A tournament on n vertices (T_n), is a directed graph obtained by assigning a direction for each edge in the corresponding undirected complete graph of n vertices (K_n)



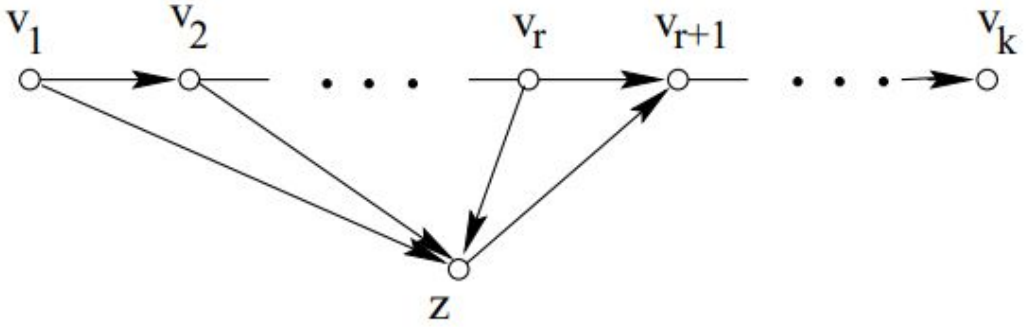
K_4



T_4

Redei's Theorem

Statement: Every tournament has a directed Hamiltonian path.

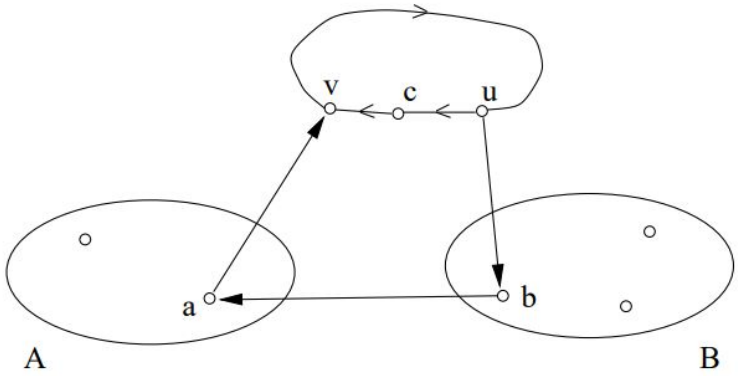


Camion-Moon Theorem

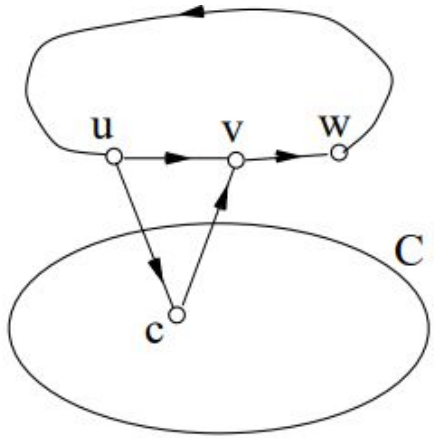
Statement: Every strongly connected tournament has a directed Hamiltonian cycle.

By induction: We assume a cycle of size k to exist and prove that k+1 exists.

Case1 : C is empty

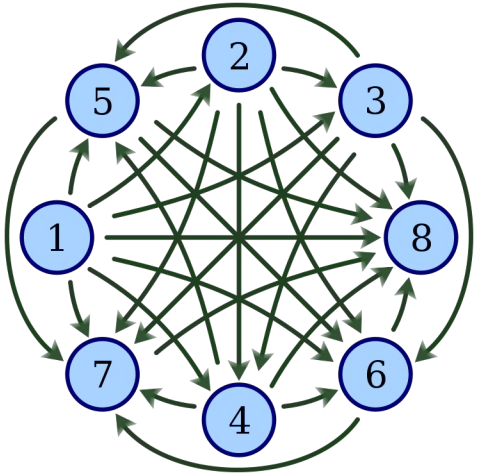


Case2 : C is Non-empty



Transitive Tournament

A tournament on n vertices is **transitive**, if ***a*** $\longrightarrow$ ***b*** and ***b*** $\longrightarrow$ ***c*** exists in the tournament, then, ***a*** $\longrightarrow$ ***c*** also exists in the tournament.



T_8

Some Properties

- Transitive Tournament does not have any directed cycles.
- There is a unique Hamiltonian path in a Transitive Tournament
- There exists a strict ordering of vertices.

Algorithms

- **Finding Hamiltonian Path in Tournament**
- **Feedback Vertex Set in Tournaments of size k**
- **Finding Minimum Dominating Set in Tournaments**

Finding Hamiltonian Path in Tournaments in O(nlogn)

Algorithm - Hell and Rosenfield (1983)

- Assume we have a path of k vertices in path: v_1-> v_2->...->v_k.
- We take an arbitrary node v_{new} to add to the path. We use a binary search methodology to find a position in the path
- We choose the middle most node in the path. Let it be v_{mid}. Now we check the edge between v_{mid} and v_{new}. This can lead to 2 cases:
- Case1 : v_{mid}<-v_{new}
  - This implies that if we check the left half of the path, it is compatible for us to place v_{new} at the end of the left half.
  - Thus we recursively search in v_1, v_2, … v_{mid-1}

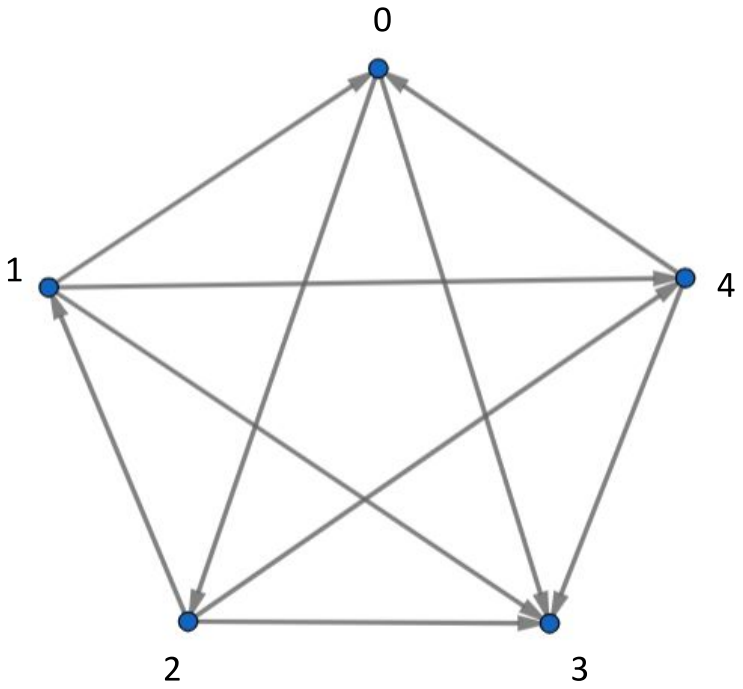
Algorithm

- Case 2: v_{mid}->v_{new}
  - This implies that it is compatible for v_{new} to be put at the left end of the right half of the path.
  - Thus we recursively search in v_{mid+1}, v_{mid+2}, … v_k
- Base Case: If the size of the path is 1 or 2 we manually check all possible insertions and choose a valid position for v_{new}.

Time Complexity and Correctness

- Time Complexity: Binary search for every k sized path from k=1 to k=n
  - Time Complexity = $\Sigma O(\log(k)) = O(\log(n!)) = n\log(n)$
- Correctness:
  - Follows from Redei's theorem.
  - For every k there will always exist a position for our new arbitrary node to fit into the path. Same holds for every k/2 path taken in the binary search recursion.
- Can we do better?
  - Answer : No.
  - Intuition : In transitive tournaments (Subset of tournaments), finding a hamiltonian path is equivalent to sorting.

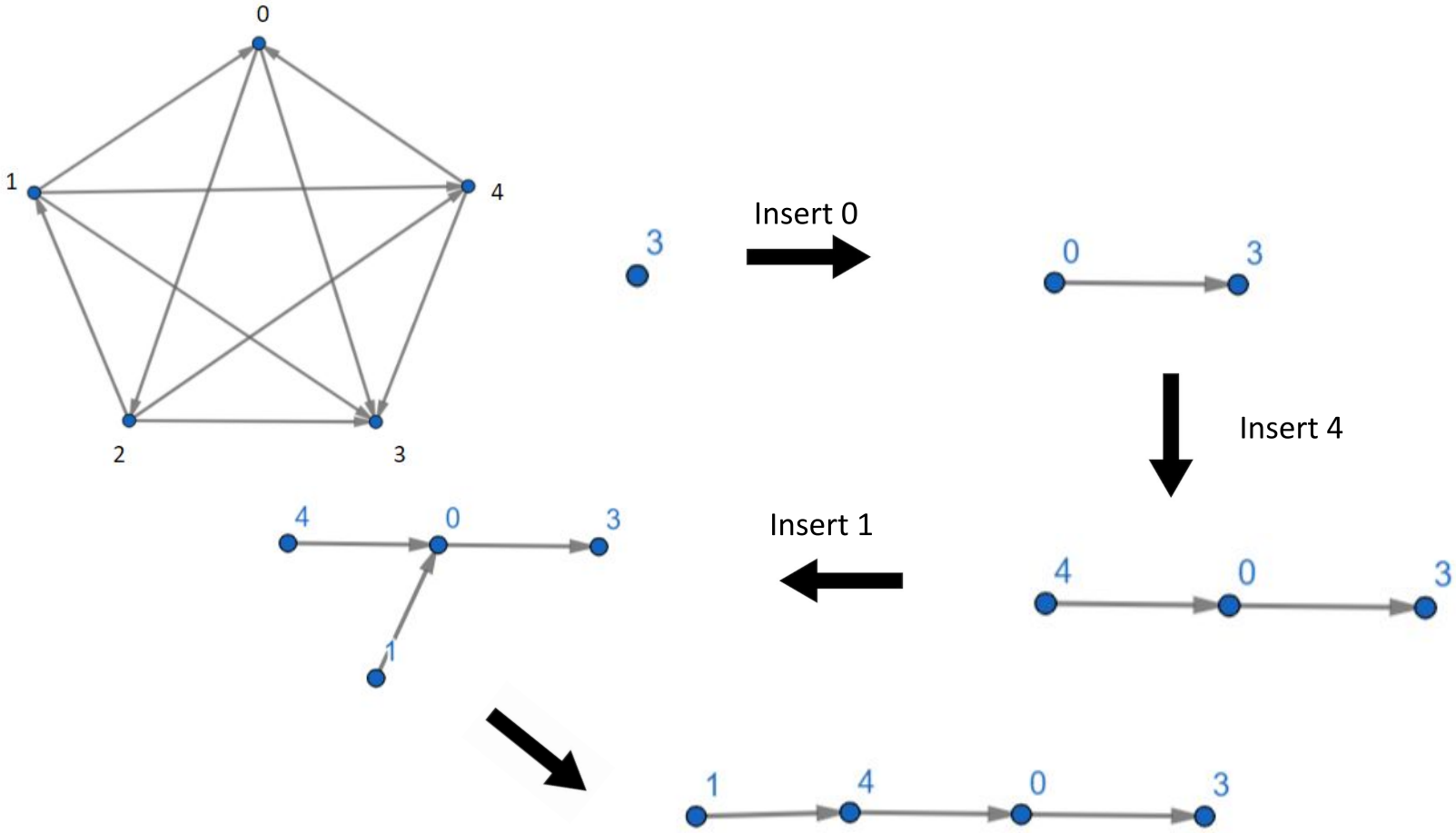
Code Implementation and Example



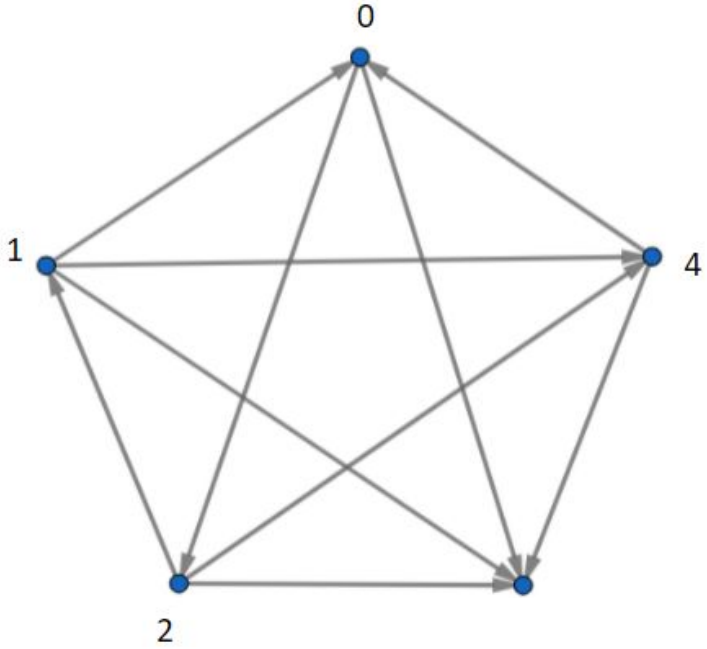
Adjacency Matrix:

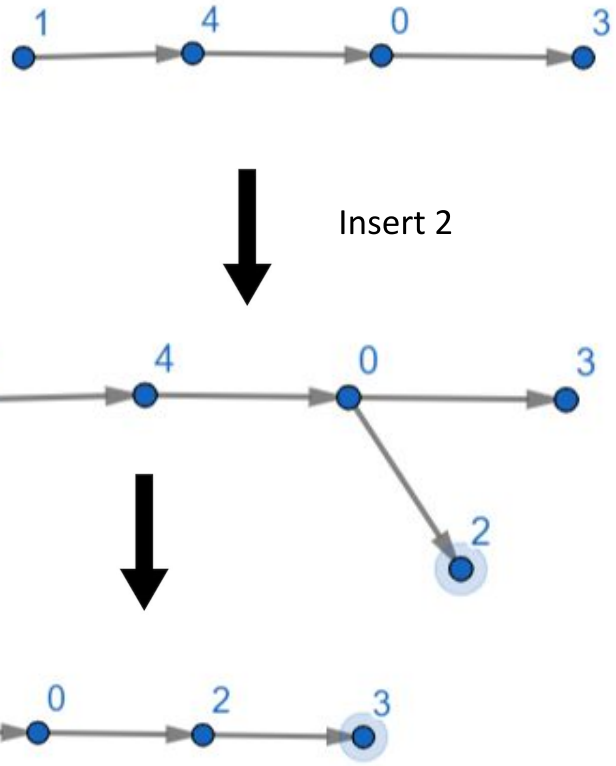
0 0 1 1 0
1 0 0 1 1
0 1 0 1 1
0 0 0 0 0
1 0 0 1 0

Order of Insertion of Nodes:
3, 0, 4, 1, 2

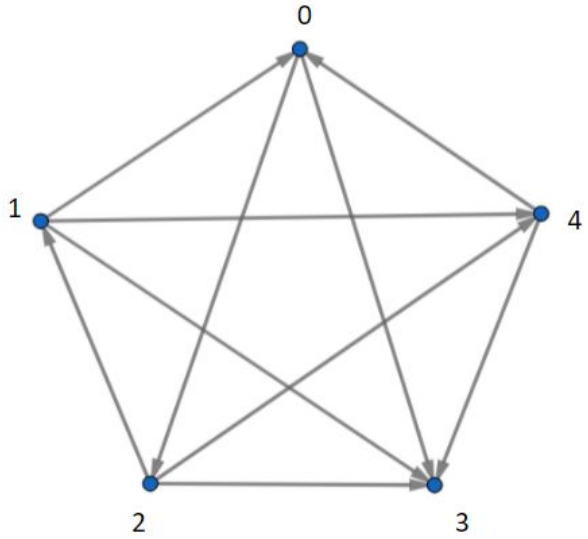

0
1
4
2
3
3
Insert 0
0
3
Insert 4
4
0
3
Insert 1
4
0
3
1
1
4
0
3

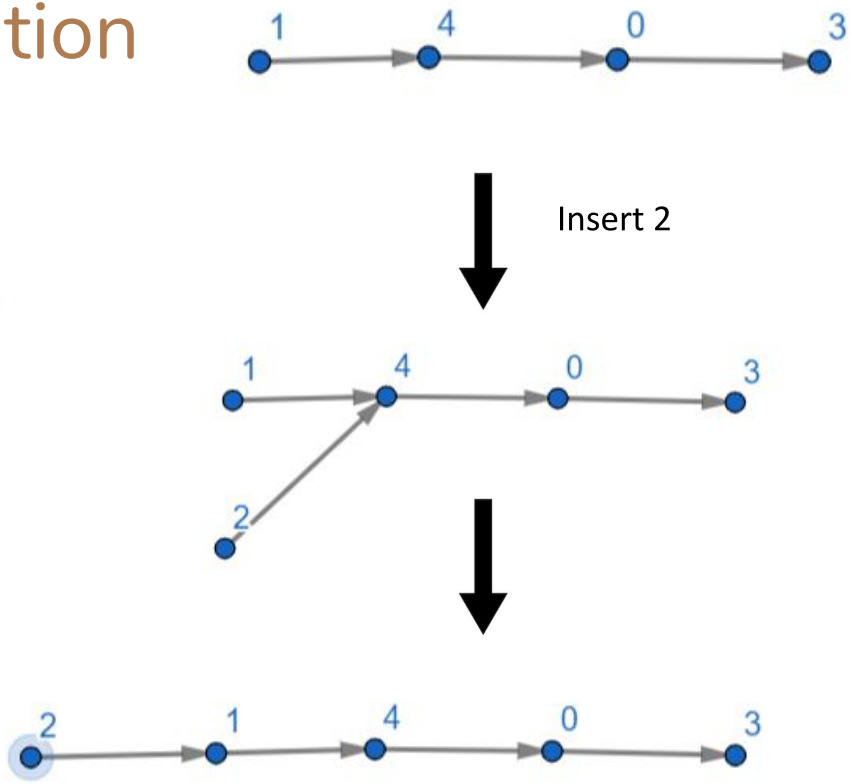
Final Solution





Alternate Solution





Sensitivity of Hamilton path to edge reversal

- We observe that reversing an edge in a tournament graph, the graph still remains a tournament.
- If the edge reversed does not lie on the hampath, then the this hamilton path will also be a hamilton path for the new graph.
- Now if an edge of a Hamilton path of a tournament is reversed, we show the following properties:
    - A Hamilton path of the new graph can be found using the older hamilton path in O(n) time.
    - The subparts of the older Hamilton path divided by the reversed edge maintain the same individual ordering in the new hamilton path.

Finding the New Hamilton path

- Let the ordering of vertices $v_1, v_2, \ldots v_n$ be a hamilton path in a tournament of n vertices. Let us consider that the edge between v_r and v_{r+1} is reversed.
- Now we have two portions of the older hamilton path :
  - P1 : $v_1, v_2, \ldots v_r$
  - P2: $v_{r+1}, v_{r+2}, \ldots v_n$
- Now we iterate with two pointers which start from the end of P1, P2 individually.
- First we move from v_n backwards in P2 looking for a position to insert v_r from P1.
- Once we find this location we insert v_r and then continue the procedure for v_{r-1}
- Observe that since we know there exists an edge v_{r-1}->v_r we can continue from there to insert v_{r-1} without any incompatibility issues.
- We continue until all nodes in P1 are inserted into P2, finally P2 becomes our new Hamiltonian Path. Every node is iterated through once and hence the algorithm is O(n).

Continued

- We know that P1 and P2 originally maintain different portions of the older Hamiltonian path. While generating the newer Hamiltonian path we do not disturb the ordering of P2's vertices, and we continually insert vertices from P1 from v_r to v_1. Thus we also maintain the ordering of vertices in P1.
- Hence we have shown that the final hamiltonian path consists of all the vertices from P1 and P2 maintaining their individual ordering.

Some Further Results

- The paper with the above method also went on to show that any generalized hamiltonian path for a tournament T_n follows the time complexity : $O(n\log^k(n))$ where k is a parameter of the required sequence. k = [0,1]
- k=1 for directed hamiltonian path (Proof for which was given earlier).
- k=0 for anti-directed hamiltonian path.
  - $v_1 \to v_2 \leftarrow v_3 \ldots \to v_n$

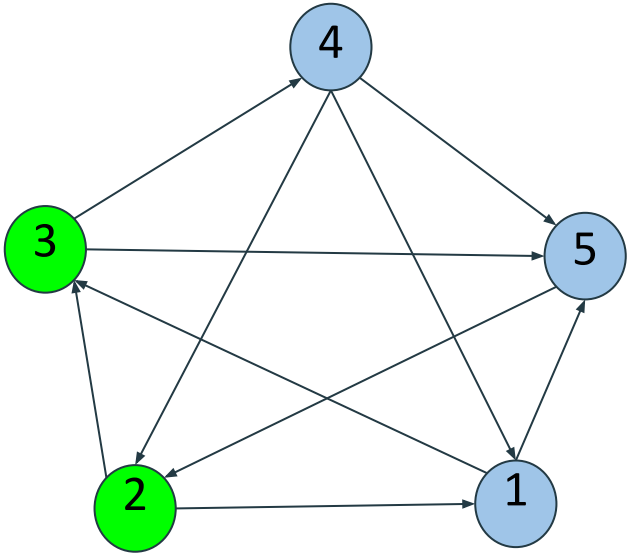
Finding Feedback Vertex Set of size k in Tournaments

Problem

Given a Tournament **T** and an integer ***k***, find a vertex set **X** of size at most ***k*** such that **T - X** is a Transitive Tournament in $O(2^k n^2 \log n)$ time.

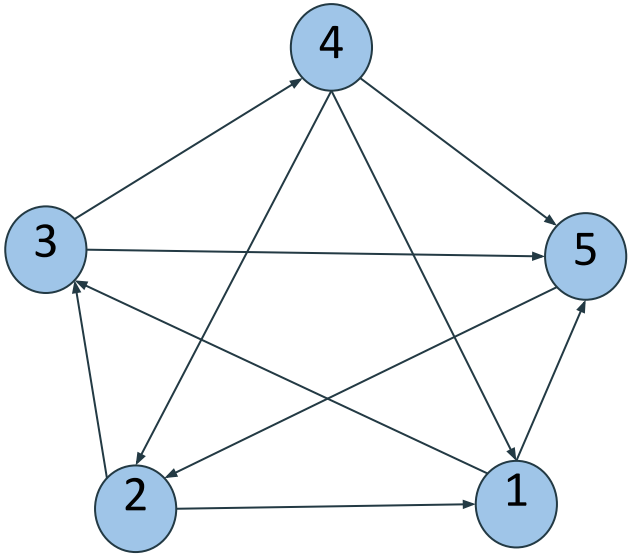
N = 5, k =2

FVS = {1, 2}



N = 5, k =1

No Solution



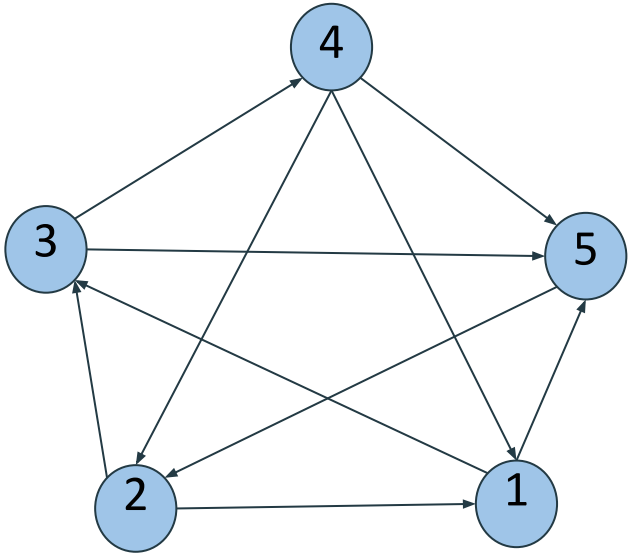
Main Idea

- Here, we use the algorithmic technique of iterative compression

```
1: function ITERATIVE-COMPRESS(T)
2:     V' ← φ
3:     X ← φ
4:     for all v ∈ V(T) do
5:         V' ← V' ∪ {v}
6:         X ← X ∪ {v}
7:         X ← COMPRESS (T[V'], X)
8:     return X
```

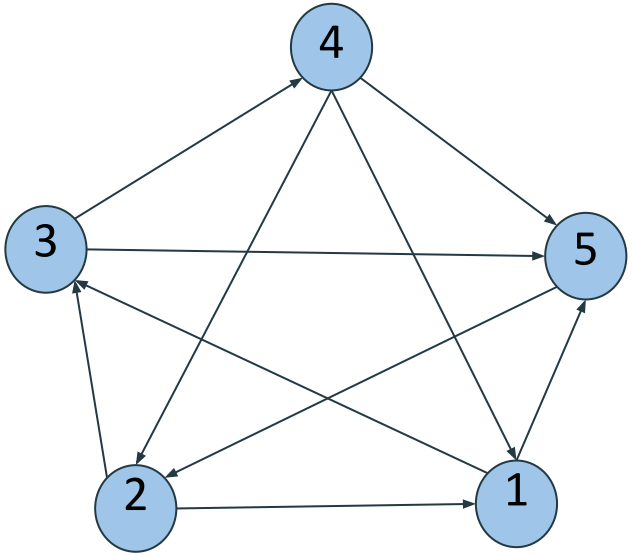
- We increase the instance size step by step, calculating the smallest Feedback Vertex Set (FVS) for that instance.
- Loop Invariant: **X** is a minimum FVS for **T[V']**

Step - 1



- **Add vertex 1 to T**

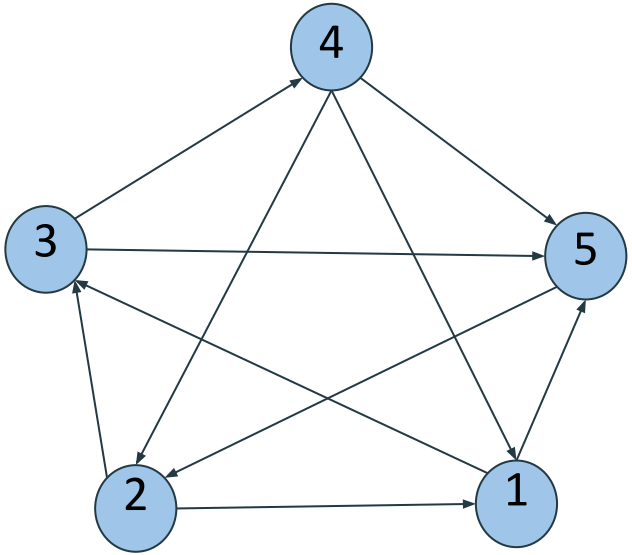
Step - 2

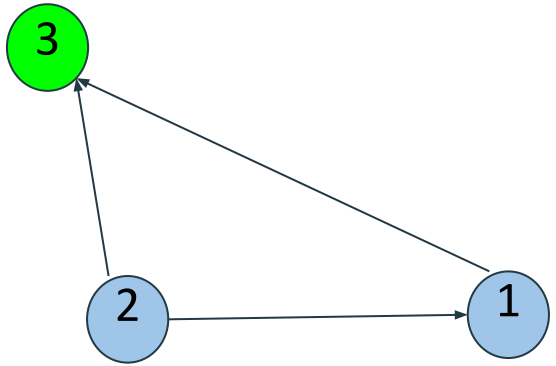


- **Add vertex 2 to T**

Step - 3

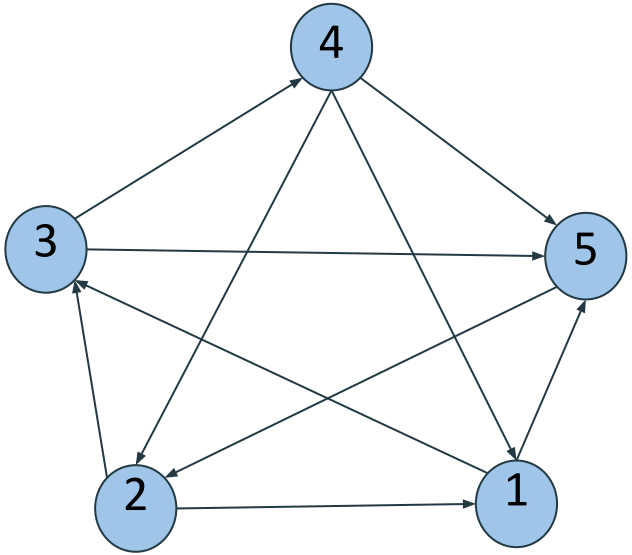
- Add vertex 3 to T
- Add vertex 3 to FVS

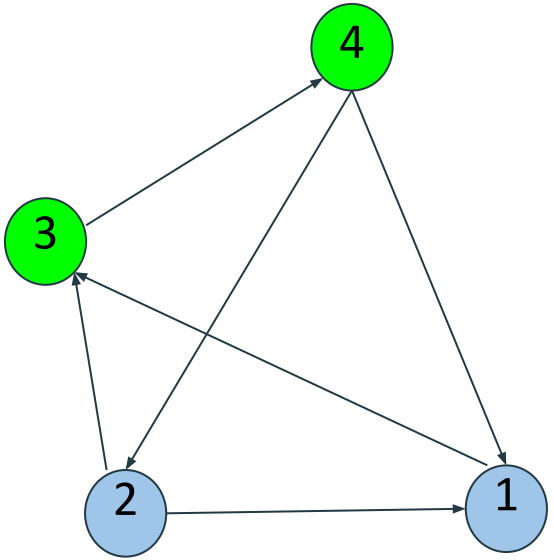




Step - 4

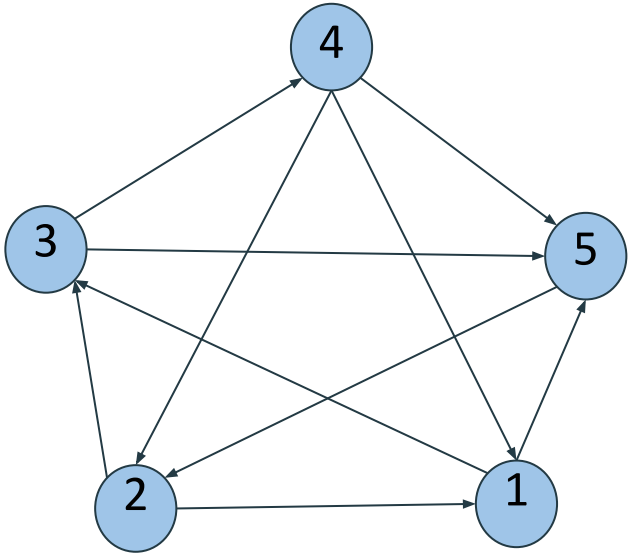
- Add vertex 4 to T
- Add vertex 4 to FVS

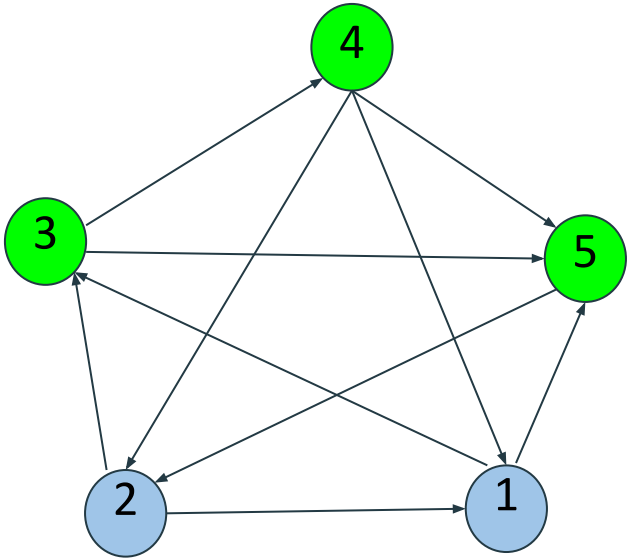




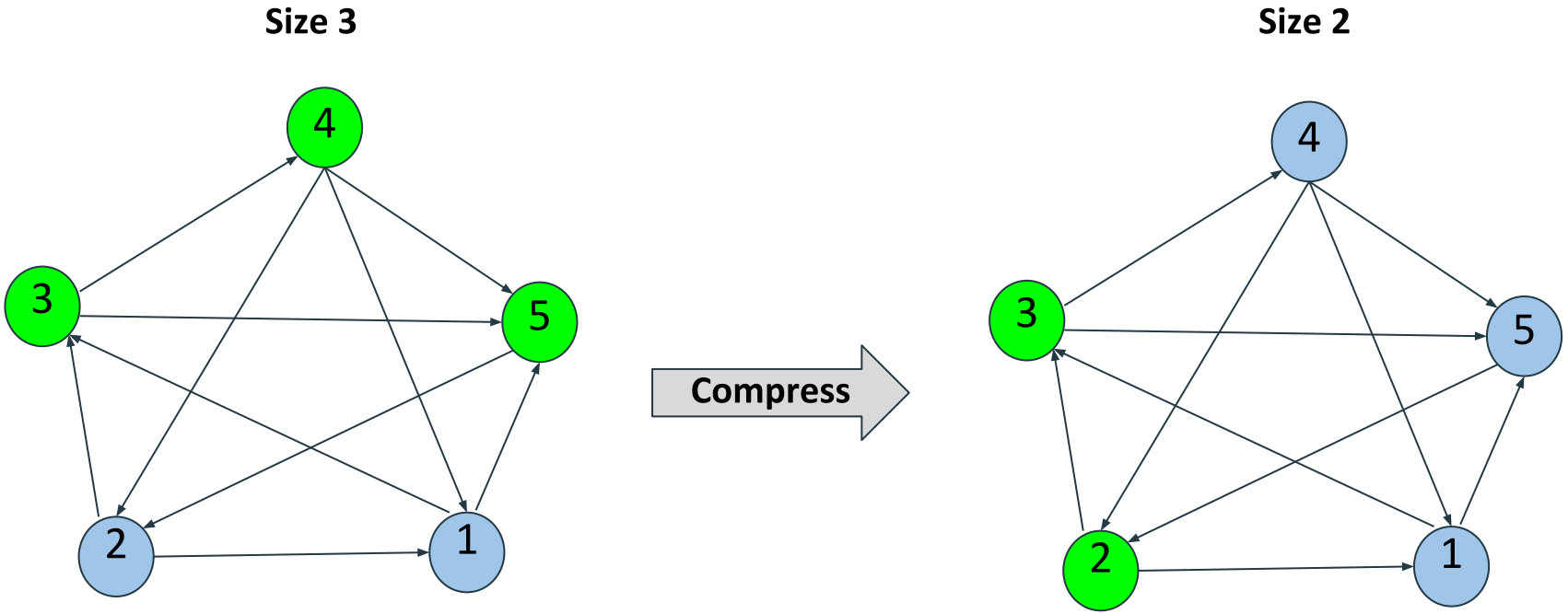
Step - 5

- Add vertex 5 to T
- Add vertex 5 to FVS
- Compress





Step - 5 (contd.)



Compress Routine

- This subroutine aims to decrease the size of the solution.

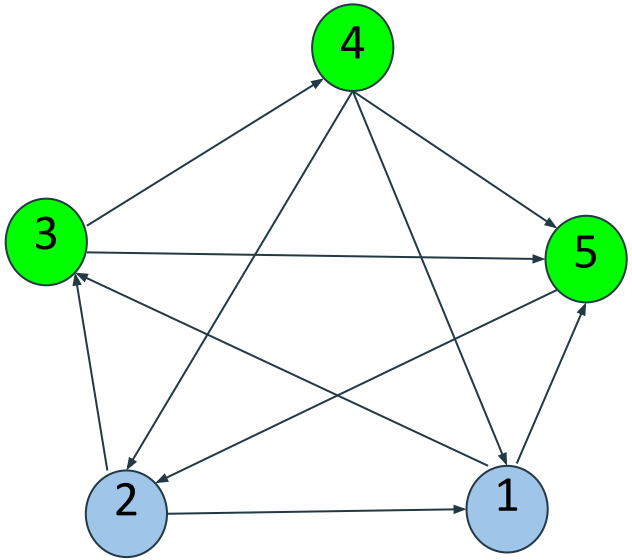
```
1: function COMPRESS(T, X)
2:     for all S ⊆ X do
3:         Z ← X − S
4:         if T[S] is transitive then
5:             T′ ← T[V(T) − Z]
6:             S′ ← DISJOINT-COMPRESS (T′, S)
7:             if |S′| < |S| then
8:                 return S′ ∪ Z
9:     return X
```

- We try all possible subsets of **X** that intersects with the optimal solution.

X = {3, 4, 5}

S = {4, 5}

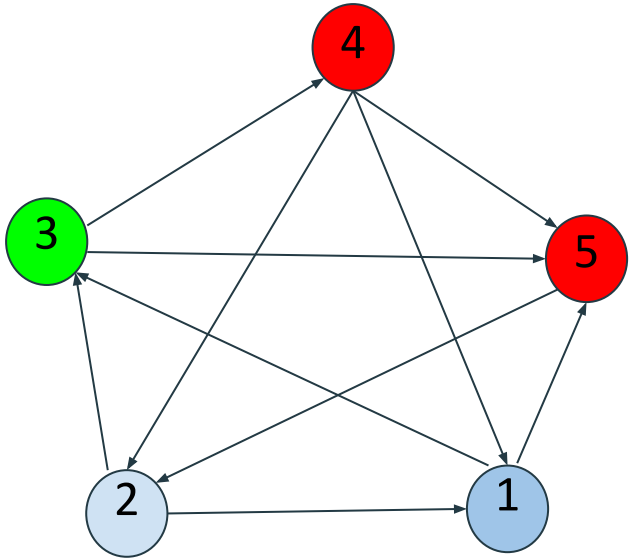
Z = {3}



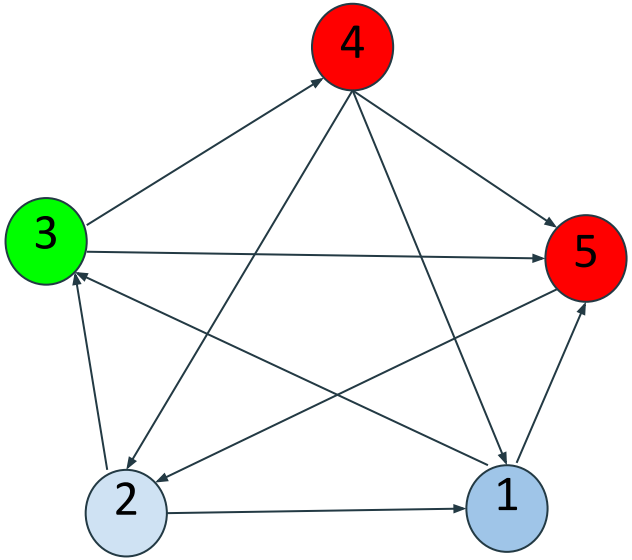
X: Original FVS

S: Vertices not in optimal FVS

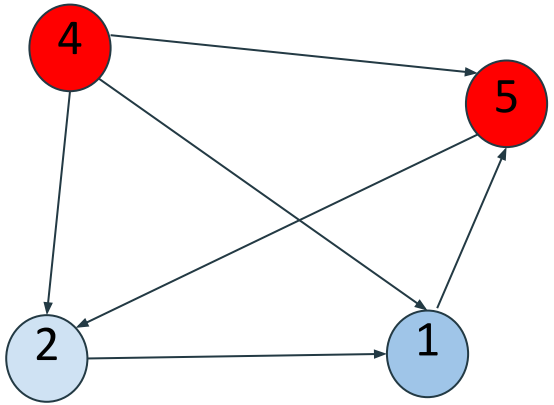
Z: Intersection with optimal FVS



Step - 1

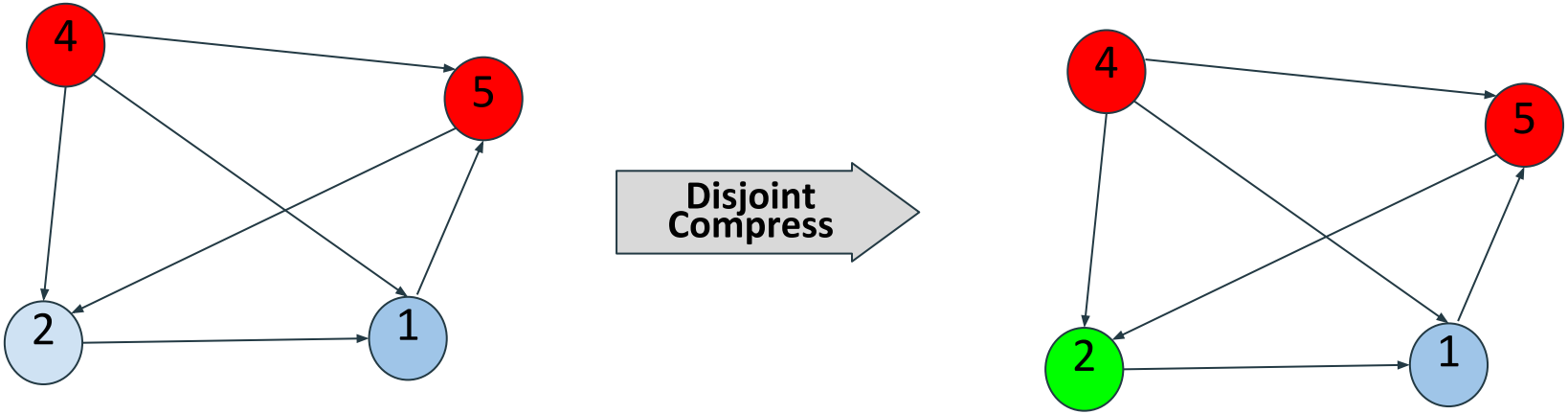


- Remove Z from instance



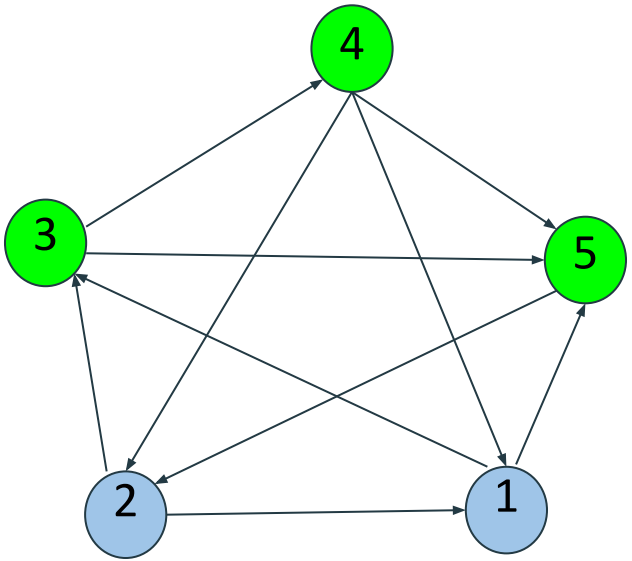
Step - 2

- Find optimal FVS disjoint from S using Disjoint Compress subroutine

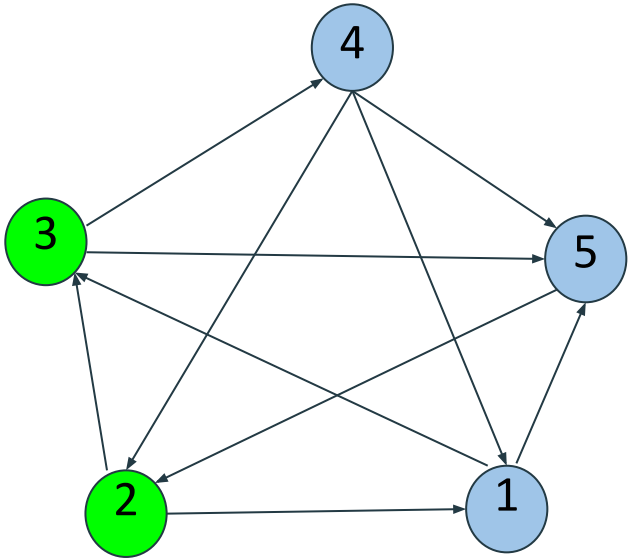


Step - 3

X = { 3, 4, 5}



- S’ = {2}
- Z = {3}
- FVS = S’ ∪ Z
- Return FVS (if better than X)

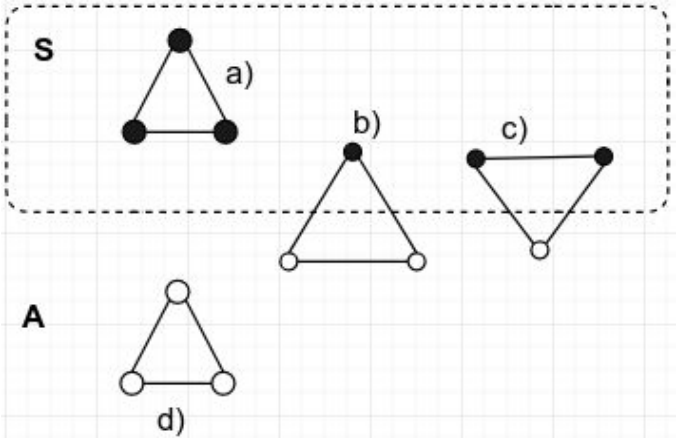


Disjoint-Compress

- This subroutine tries to find an optimal FVS **X,** disjoint from given FVS **S,** for tournament **T.**
- Easier to solve, gives an extra underlying structure to the problem instance.
- **Claim 1**:- A Tournament is a Directed Acyclic Graph (DAG) if and only if it doesn't contain any directed triangles.
- **Claim 2**:- A Transitive Tournament has a unique total ordering $\prec$, such that if $a \longrightarrow c$ exists, then $a \prec c$.

Disjoint-Compress

- **Four** types of triangles possible (in general).



A = V(T) - S

- Triangles **a** and **d** not possible.
- In triangle **c**, the vertex in **A** must be included in FVS

Disjoint-Compress

- π_A and π_S are the topological orderings of **T[A]** and **T[S]**.
- $\forall v \in A,\ T_v = T[S \cup \{v\}]$ is transitive (acyclic), and hence have unique total ordering (v is inserted in π_S).
- **P(v)** is position of v in the topological ordering of T_v. Create new ordering of **A,** π_X . $\pi_X(u) < \pi_X(v)$ iff either:-
  - ➔ **P(u) < P(v)**
  - ➔ **P(u) = P(v)** and $\pi_A(u) < \pi_A(v)$

Disjoint-Compress

- **Claim 3:- T[S ∪ B] (B ⊆ A)** is acyclic, iff vertices of B form a common subsequence of π_A and π_X .
- Maximum vertex set **B ⊆ A**, we can include in a transitive tournament is the set of vertices from LCS of π_A and π_X.
- Minimum FVS **X**, disjoint from **S**, is **X = A - B**.
- Disjoint-Compress runs in O(nlogn) + O(nk) = O(nlogn)
- Total algorithm runs in $O(n).O(2^k).O(n\log n) = O(2^k n^2 \log n)$

Finding Minimum Dominating Set in Tournaments

Problem

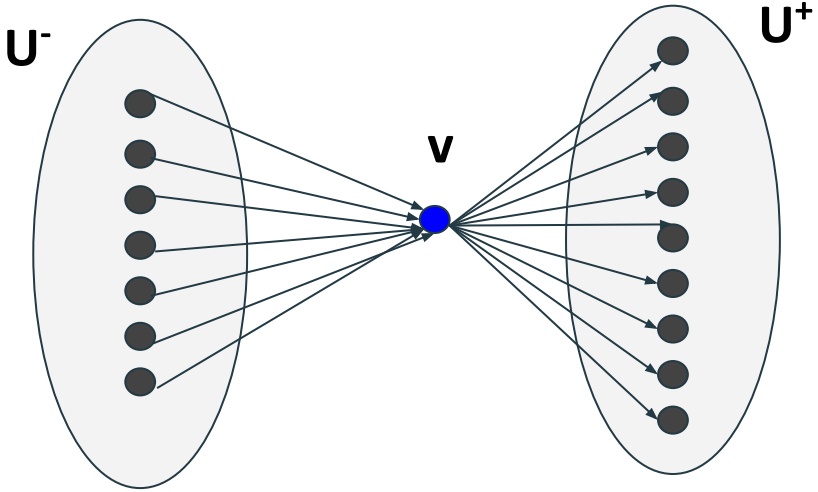
Given a Tournament **T**, find a vertex set **X** of minimum size such that for every vertex **v** in **T - X**, $\exists\, \mathbf{v_0} \in \mathbf{X}$, such that $\mathbf{v_0} \rightarrow \mathbf{v}$ exists, in time $n^{O(\log n)}$

Main Idea

- We will use the fact that every tournament T_n, has a dominating set of size at most **1 + logn**

Definitions

- For a vertex **v** in **T**, let U^- and U^+ denote the set of in-neighbours and out-neighbours of **v,** respectively.
- U^- and U^+ form a partition of **V(T) - {v}**



Proof (by induction)

- We will prove this by induction on **n**. For **n = 1**, trivial.
- Since the sum of all out degrees in $\mathbf{T_n}$ is **n(n-1)/2**, then there exists a vertex **v** with out degree at least **(n-1)/2** .
- Then for that vertex **v**, $|\mathbf{U^+}| \geq \mathbf{(n-1)/2}$ and $|\mathbf{U^-}| \leq \mathbf{(n-1)/2}$. Pick this vertex in the dominating set. $\mathbf{U^+}$ is dominated.
- Now, we consider tournament $\mathbf{T[U^-]}$. By, induction hypothesis, there is a dominating set of size **logn** for $\mathbf{T[U^-]}$.
- Therefore, there exists a dominating set of size at most **1 + logn** for $\mathbf{T_n}$

Algorithm

- Now we can brute force all possible sets **X** of size **1** to **1 + logn**.
- As soon as we find a dominating set, we stop.
- Time Complexity :- $\mathbf{O({}^{n}C_{1+\log n}) = O(n^{O(\log n)})}$

Statistics on Size of Minimum Dominating Set*

N	Avg.	Min.	Max.	Limit
5	1.656	1	2	3
8	1.953	1	2	4
12	2.009	1	3	4
16	2.148	2	3	5
20	2.385	2	3	5
23	2.604	2	3	5
27	2.802	2	3	5
32	2.929	2	3	6

* Generated min(2^n, 1000) random tournaments

References

- Wikipedia. Tournament (graph theory). https://en.wikipedia.org/wiki/Tournament_(graph_theory)
- Cygan, Marek, et. al. Iterative Compression. Parameterized Algorithms. Springer. 2016.
- Cygan, Marek, et. al. Fixed-parameter Intractability. Parameterized Algorithms. Springer. 2016.
- Moon, J. W. (1966), "On subtournaments of a tournament", Canadian Mathematical Bulletin, 9 (3): 297–301, doi:10.4153/CMB-1966-038-7.
- Bar-Noy, A.; Naor, J. (1990), "Sorting, Minimal Feedback Sets and Hamilton Paths in Tournaments", SIAM Journal on Discrete Mathematics, 3 (1): 7–20, doi:10.1137/0403002.
- Pavol Hell, Moshe Rosenfeld (1983), “The complexity of finding generalized paths in tournaments”, Journal of Algorithms: Volume 4, Issue 4, doi:10.1016/0196-6774(83)90011-1

THANK YOU!