Algorithms and Data Structures for Computational Biology

Topological sorting

(Slides credits: these slides are a revised version of slides created by Prof. Claudio Sacerdoti Coen)

Prof. Ivan Lanese

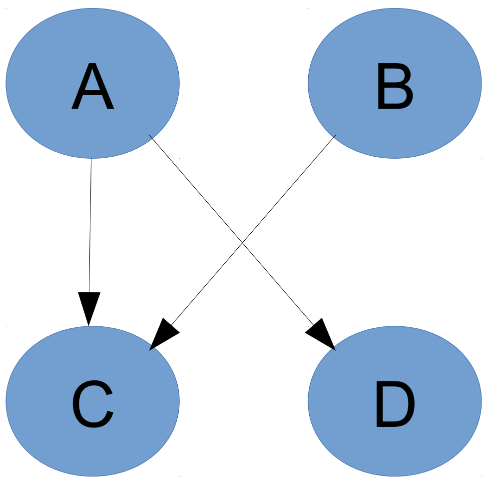
DAGs as Dependency Graphs

- A dependency graph is a graph representing causal dependencies between tasks
  - "I need to put my shoes on before going out" can be represented as a direct edge from vertex "put my shoes on" to vertex "going out"
- In general, tasks to be completed can be arranged into DAGs (Direct Acyclic Graphs)
  1) vertices are tasks
  2) an edge (v,w) means that v must be completed before w (or is required by w or ...)
- Question : can we use a general directed graph instead?
- Exercise : draw the graph to cook your favourite meal

Topological sorting

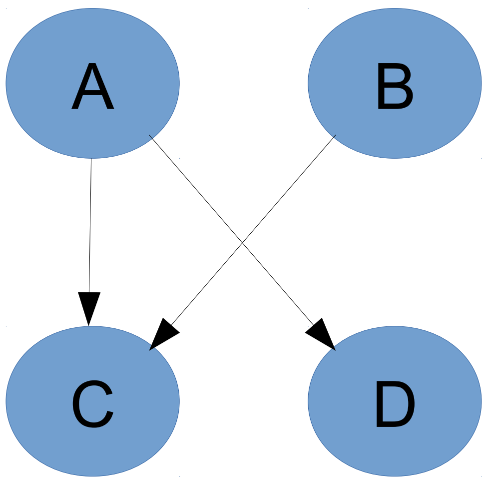
- Given a DAG (V,E), (v1, …, vn) is a topological sorting of the
- nodes in V iff:
  1) {v1, …, vn} = V
  2) for each i < j, there is no path from vj to vi
- Tasks can be executed in topological order without violating dependencies
- Question : can a DAG be topologically sorted in two different ways?

Topological sorting: example



- { A, B, C, D }, { B, A, D, C }, { A, D, B, C } are all valid topological orders
- Question: are there more?

Topological sorting algorithm ideas



- Topological sorting algorithm – ideas :

1) nodes are added one by one in front of the result list
2) a node is to be added only after adding all nodes that are reachable from it

- Question: how to know what nodes are reachable from a given one?

Topological sorting algorithm

- TopoSort(G)

```
    L = empty_list()
    for u in vertices(G)
        modified_dfs(G,u,L)
    return L


modified_dfs(G,u,L)
    mark(u)
    for v in AdjSet(G,u)
        If not marked(v)
            modified_dfs(G,v,L)
    add(L,v) // v is added to L after all nodes reachable from v
```

Exercise

- Run the previous algorithm on the graph of Slide 4 multiple times, changing only the order in which vertices(G) returns the nodes