Algorithm Assignment 6

10185101210 陈俊潼

22.1-6

Observe a adjacency matrix with a universal sink:

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

We found that when a vertex is an universal sink,

Let `M` be the adjacency matrix of size [V, V]. Start from point `M[i, j]`.

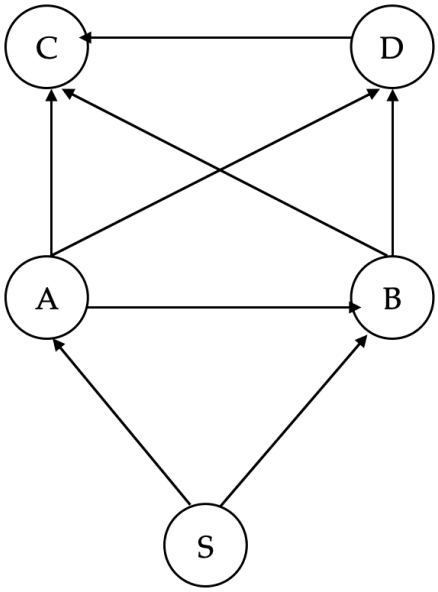
- If `M[i, j] == 0`, then check `M[i, j+1]`
- If `M[i, j] == 1`, then check `M[i + 1, j]`

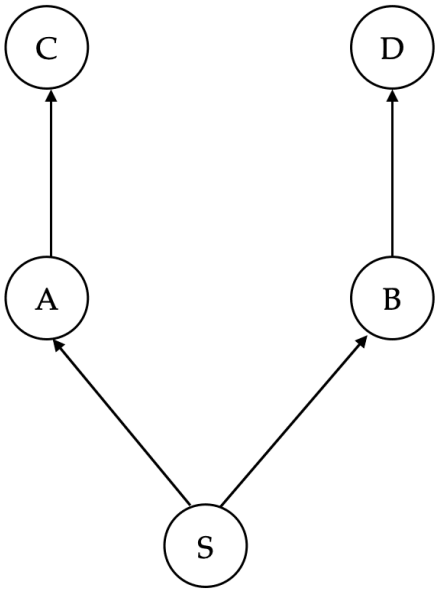
Repeat the process above, until when we found `i == V` or `j == V`. This might a universal sink. Then check if the row is all 0. If true, then this is a universal sink. If such point can't be found when `i` or `j` out of boundary, there doesn't exist a universal sink.

The complexity of this algorithm is $O(V) + 2O(V) = O(V)$.

22.2-6

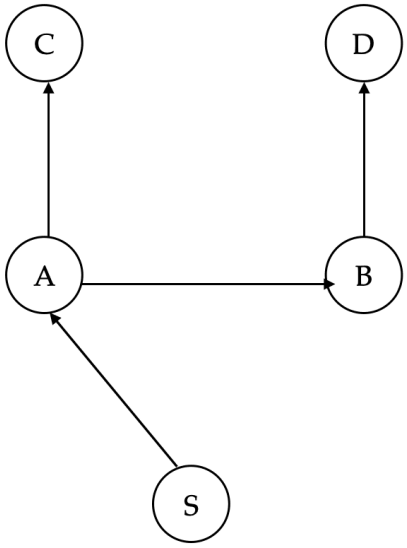
This example is shown as follows:



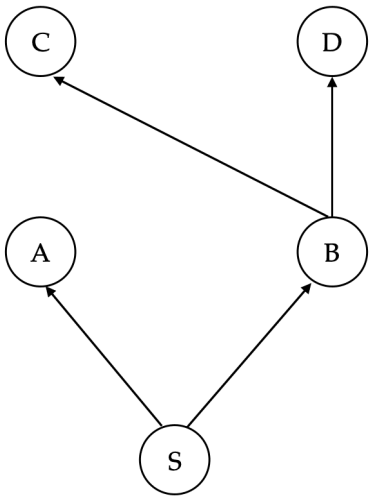


No matter how the vertex is ordered is the adjacency matrix, it's no possible to generate a tree edges as the right image.

If A first, then it will be:

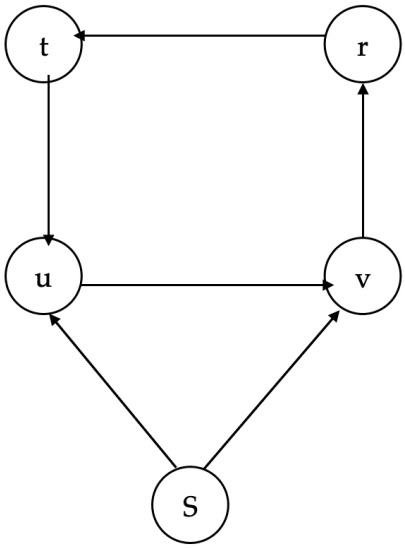


If B first, then it will be:



22.3-9

An example is as follows:



When DFS visit the above graph, if v is visited first, then the steps are:

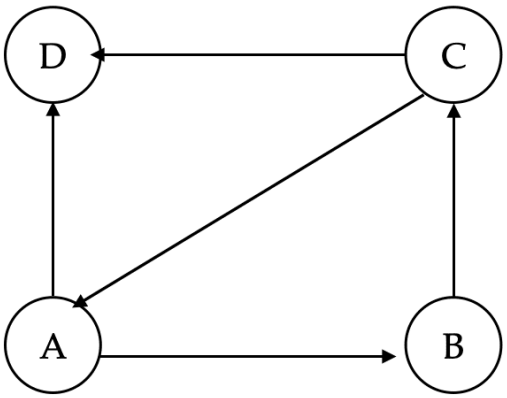
- v.d set
- r.d set
- t.d set
- u.d set
- u.f set
- t.f set
- r.f set

- v.f set

In this case, $u.f < v.d$.

22.4-4

This is not true. Consider the following case:



Use `TOPOLOGICAL-SORT(A)` on this graph, assume D precedes B.

Now that we have the order BCAD, there are two bad edges: AB, and CD. However, the best case could be ABCD. where there's only one bad edge which is CA.

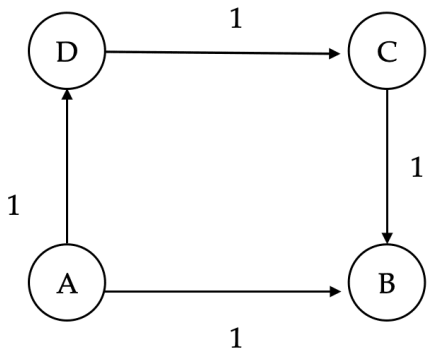
23.1-3

Because the edge (u, v) is already contained in some MST, so we can remove the edge (u, v) and then get two different MSTs. Where V_1 is the vertices of the first MST and V_2 the vertices of the second MST. In this case, we can cut the tree into two separate the tree into T_1 and T_2. The edge (u, v) is then the light edge across T_1 and T_2.

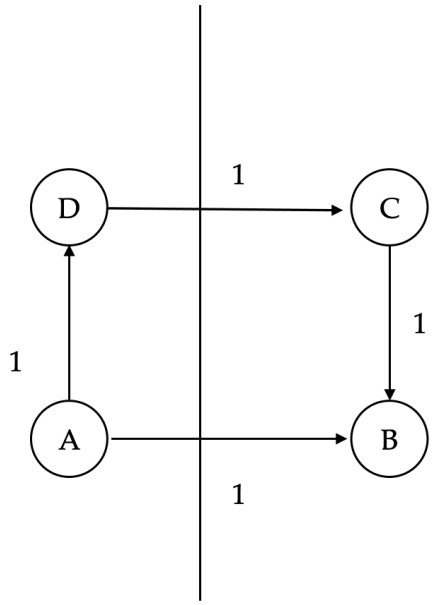
To further verify this, we assume edge (u, v) is not the light edge. So there would be another edge E, which is the light edge. But in this case, the former MST would be V_1 and V_2 and an edge E connecting them, contradicting to our given MST. So (u, v) must be the light edge.

23.1-4

The example is shown as follows:



Here for all the edges (AB, BC, CD, DA), there exists a cut (S, V-S) that (u, v) is a light edge. For example, for edge A->B:



The cut is {AD, BC}, here AB and DC are both light edges.The situation is the same for edge AD, BC, and CD. Hence the edge set will be all four edges, which of course, is not a minimum spanning tree of the original graph.

23.2-4

For the first case, we can use a van Emde Boas tree to improve the time bound to $O(E \lg \lg V)$. Comparing to the Fibonacci heap implementation, this improves the asymptotic running time only for sparse graphs, and it cannot improve the running time polynomially. An advantage of this implementation is that it may have a lower overhead.

For the second case, we can use a collection of doubly linked lists, each corresponding to an edge weight. This improves the bound to $O(E)$.

23.2-4

If the weights of the graph is in the range from 1 to |V|, then first we can use counting sort to sort all the edges, gives the complexity of $O(V + E)$. Then using the Kruskal's algorithm, it will cost $O(V + E + V \lg V)$.

If the weights of the graph is the range from 1 to W, then the different part is the time it costs in counting sort, which gives the complexity of $O(W + E)$. Then using the Kruskal's algorithn, it will cost $O(W + E + V \lg V)$ time.