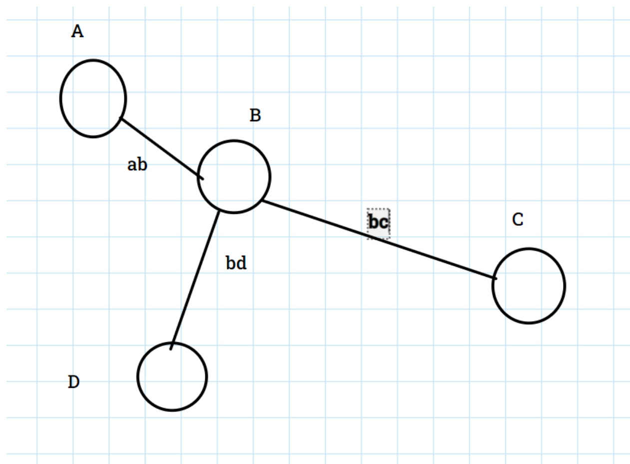
Graph:

A graph G is a collection of vertices/nodes and edges. We denote the set of nodes as V(G), and the set of edges as E(G). The edges connect between two nodes, usually representing some type of relationship between the two.

e.g.:



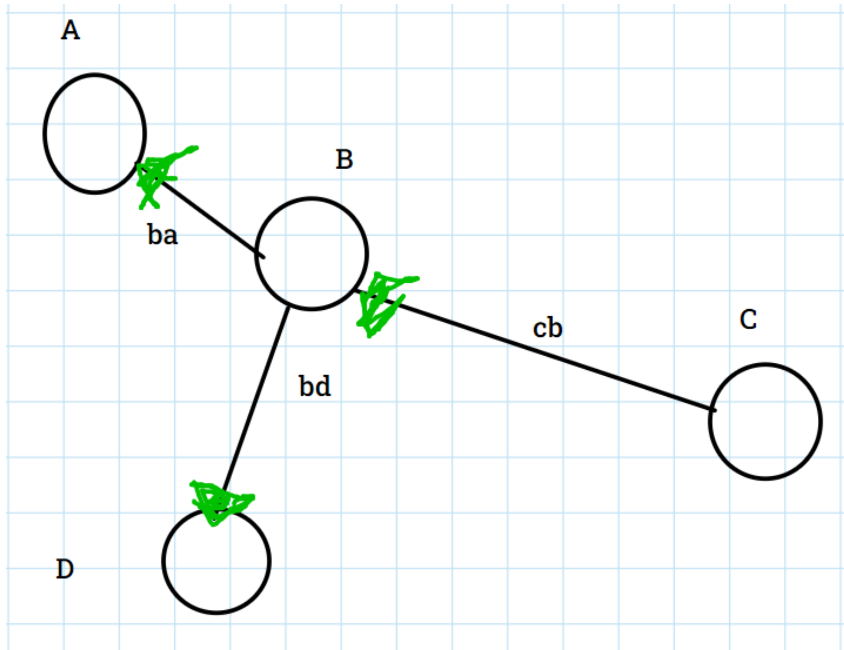
G is the collection: E(G) = {ab, bc, bd}, V(G) = {A, B, C, D}.

Digraph:

A digraph G is a graph, but each edge is known as directed. This means that it has one node it starts from, and ends at another node. Compared to undirected, this indicates a certain direction which limits what the graph represents.

For example, there can be the problem of the shortest path. In undirected graphs, as long as an edge connects two vertices A and B, then the path can go from either A to B or B to A. But, if it’s a directed graph, and it shows its direction from A to B, then a path going from B to A is not a valid path.

e.g.:



The arrows in green indicate the direction (ie, it goes from B to D) and by convention, usually the node the edge is entering goes last (bd, and not db).

Path:

A path put very simply is a way to follow edges from one node and ending up at another node.

Cycle:

A cycle is a path where only the first and last vertex are equal.

Circuit:

A circuit is a path where the first and last vertex are equal (but not necessarily the only equal pair).

Connected Graph:

A graph G is connected if there exists a path from every node in G to another node in G.

Tree:

A graph G is a tree if there exists no underlying circuit, and it is a connected graph.

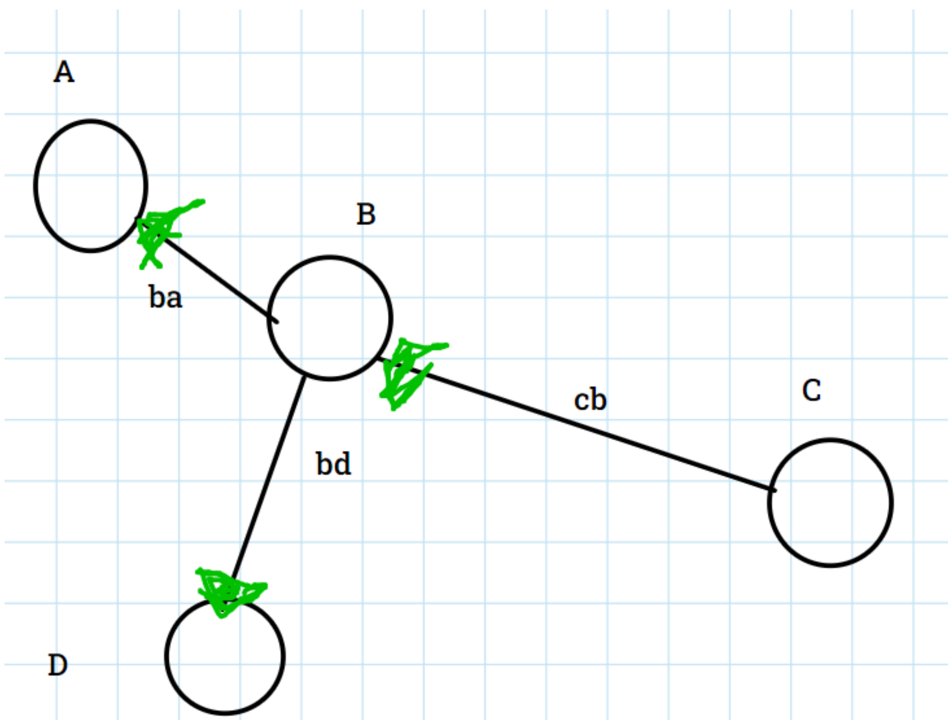
Directed Tree:

A digraph G is a directed tree, if removing the directions on the edges of G results in a tree. (Removing the directions on the edges of G is known as the underlying undirected graph).

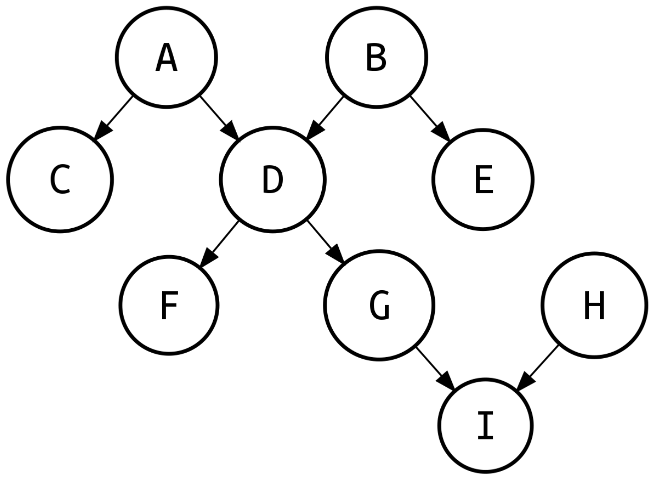
Branchings:

A branching B (also referred to as an arborescence)* is a directed tree (ie a graph G with directed edges) such that every node has at most one incoming edge (or none at all).

e.g.:



The above is a branching since no node has more than 2 incoming edges. Notice that there is exactly one node in a branching with 0 incoming edges, since it's a special type of directed tree. This node is known as the **root**.



The above is **NOT** a branching because node I has two incoming edges from G and H. It is however a directed tree because the underlying graph is a tree (no cycles/circuits).

***Author's note:** According to the Jack Edmonds talk, a branching is defined as above. But, according to Wikipedia + other sources online, a branching is a collection of arborescences, the same way a forest is a collection of trees.

Branching rooted at r:

As mentioned in the branching section, there exists exactly one node in a branching B such that there are no edges incoming. This node is known as the root, denoted by r. A branching is then rooted at this node r.

Subgraph:

A subgraph G' of G is simply a graph whose set of edges is a subset of the edges of G, and the set of nodes is a subset of nodes of G. i.e.: $E(G') \subseteq E(G)$, $V(G') \subseteq V(G)$.

Spanning subgraph:

A spanning subgraph G' of G is a subgraph whose vertex set is the same, but its edge set is a subset of the edges of G.

Branching System:

Given a digraph G, and specified root nodes r(i) (i.e. marked nodes of the graph G), a branching system is the spanning graph of G created by a collection of branchings rooted at r(i). This collection of branchings are edge-disjoint, i.e. no edge is shared amongst two different branchings that form the system.

Note that every root node has no incoming edges, and every other node must have at least one incoming edge (and at most k incoming edges given k root nodes).

Optimal Branching Systems Problem:

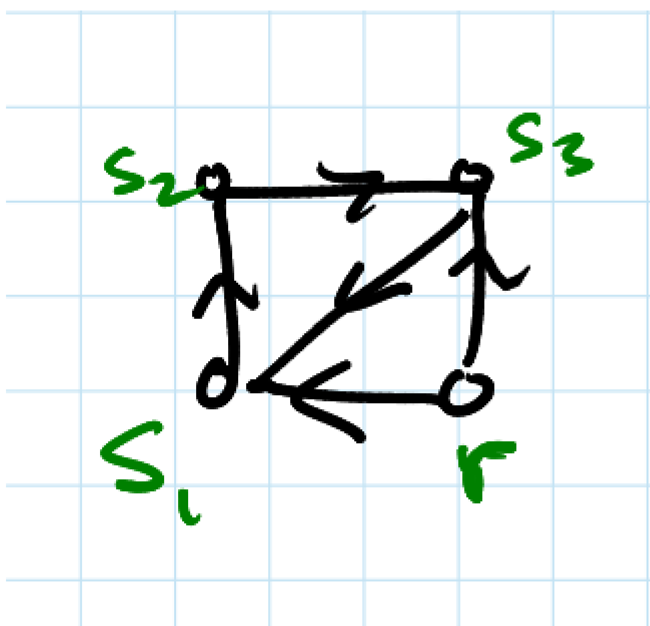
Given a digraph G, with weights for every edge, and specified root nodes r(i), what's the branching system with the least cost? (We define the cost of a branching system as the sum of the edge weights).

Notice that we can easily redefine this question in terms of max cost.

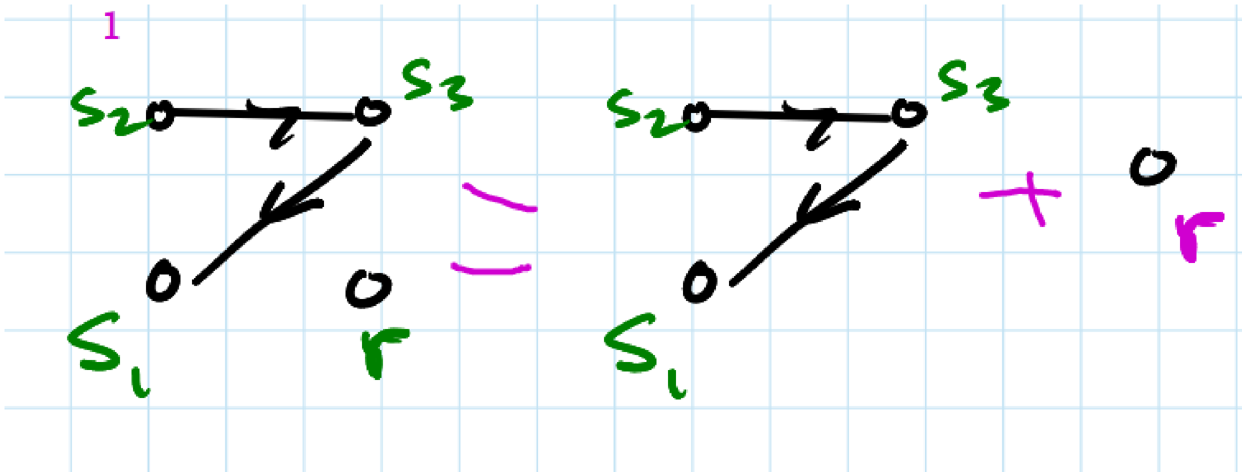
Let's explore Jack Edmonds' solution through two examples, a four node example where k = 2, and a five node example where k = 2 as well.

Four node example:

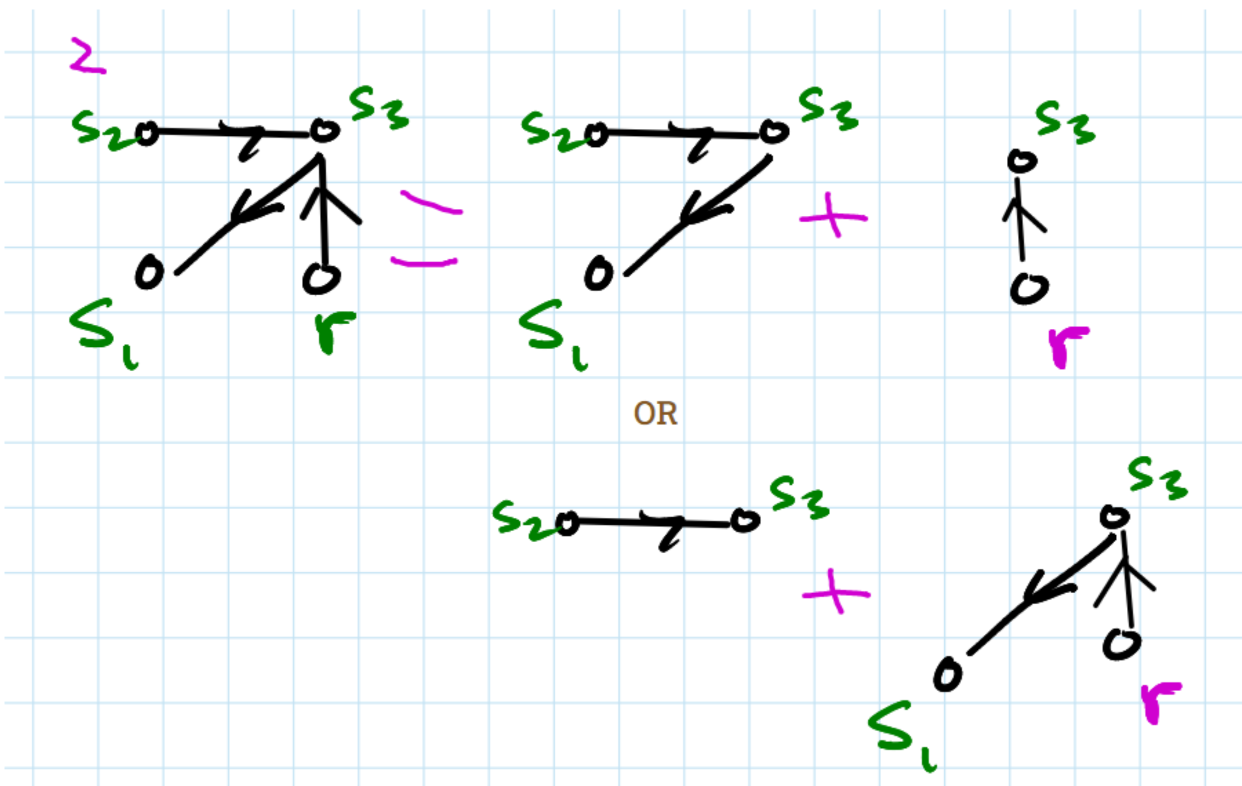
Digraph G, with root nodes r and s2:



The following are all the possible branching systems of graph G.

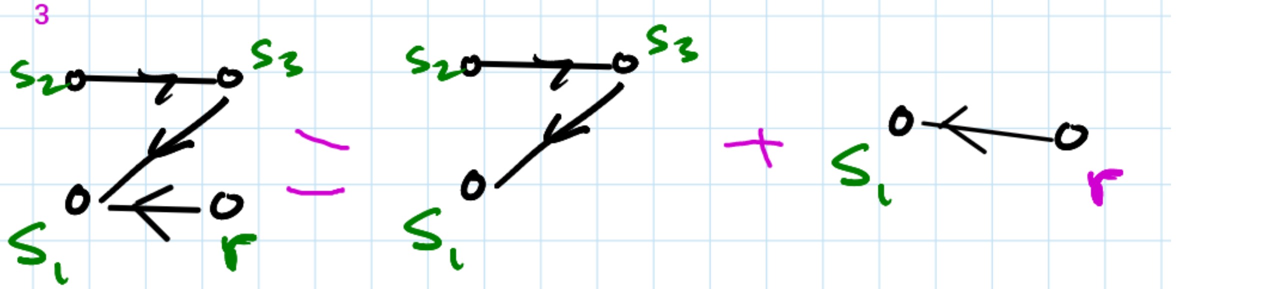


Branching System #1: Left side is the branching system, the right side are the branchings that form it.

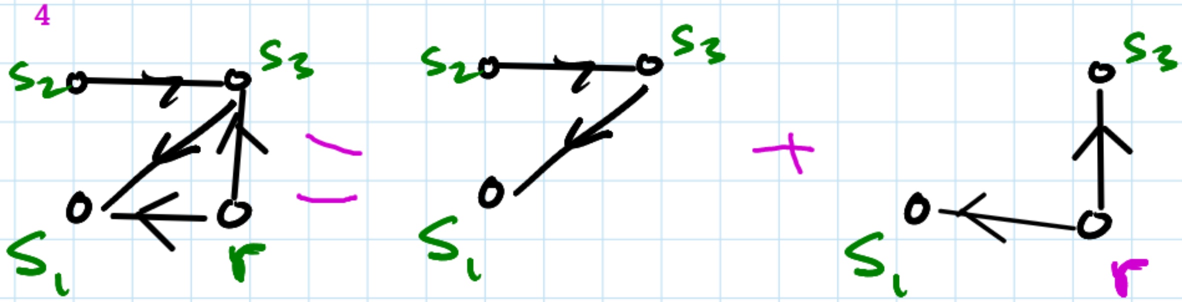


Branching System #2: What's interesting to note here is that there are actually two different underlying branchings that can form the system.

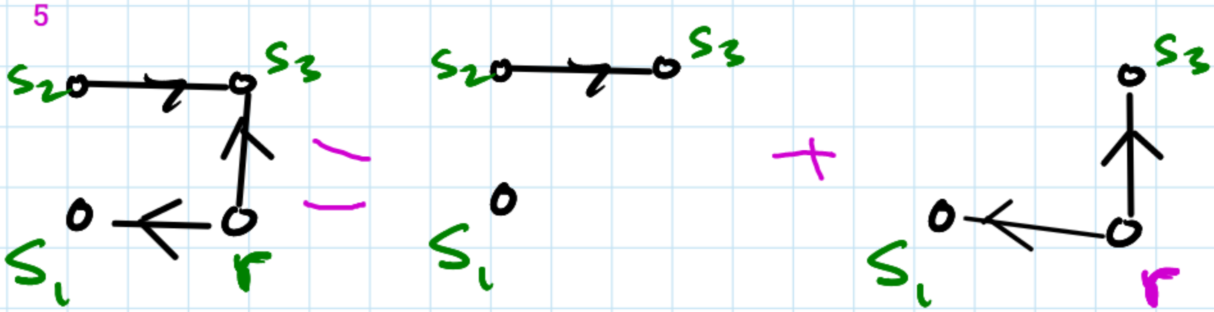
3



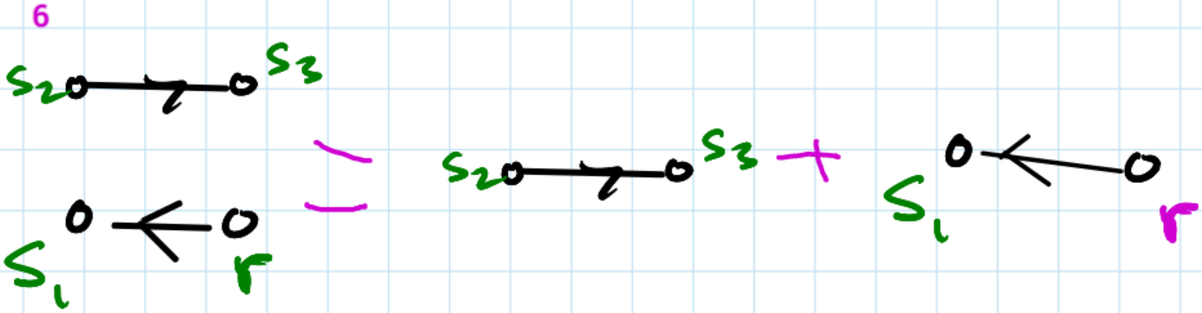
4

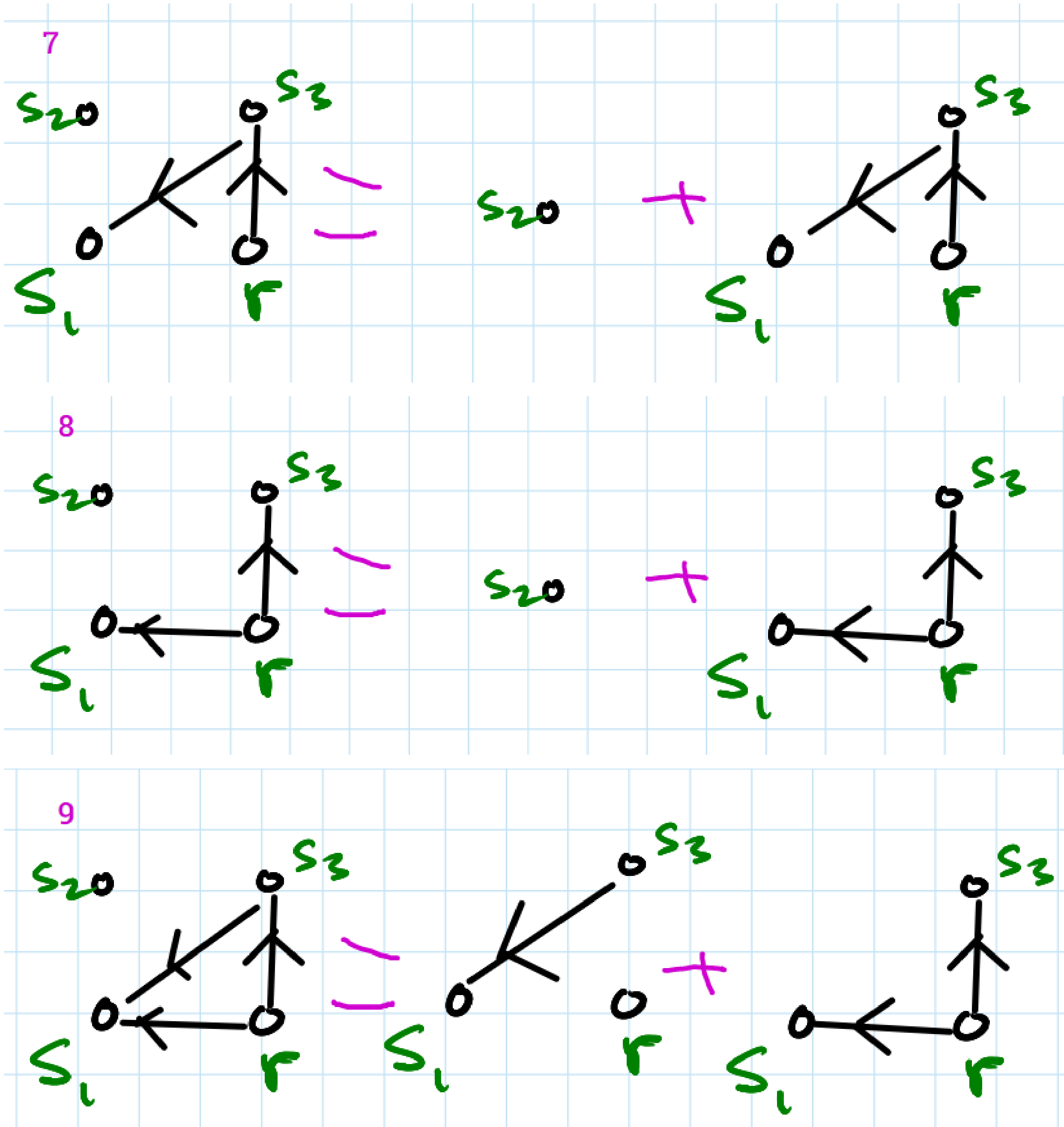


5



6





Branchings #5-9.

How should we generate these branchings? Jack Edmonds has a solution that uses the intersection of matroids. The matroids are defined as follows:

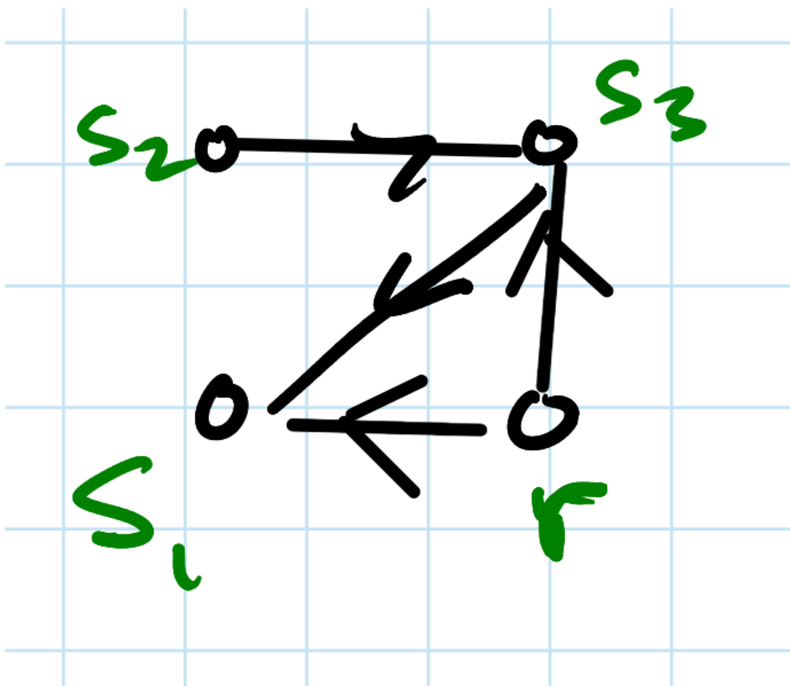
M1:

The first matroid M1 is defined that every element in M1 is a set of edges, such that none of them are directed at any root node, and at most k of them are directed toward any other node, where k is the number of root nodes.

Why does this help solve the problem? Well, notice that every branching system with roots nodes r(i) has no incoming edges into any root node. But also, it combines k branchings, and so since each branching has at most one edge into every other node, then each node must have at most k edges incoming. ****Problem: M1 is always a power set over some edges - it's exponential****

For the four node example:

M1 is defined as the power set (all combinations of edges) of the following graph:



Only the edge from S1 to S2 was removed because S2 is a root node.

M2:

As defined by Jack Edmonds, a set J of edges is in M2 when it can be partitioned into edge sets of at most k edge-disjoint forests in G. What does this mean?

Well here's an equivalent definition that's probably easier to understand:

(1) A set J of edges is in M2, when there exists a partitioning such that each subset in the partition describes a forest**, (and is edge-disjoint with all other subsets of the partition) and there are at most k subsets in the partition.

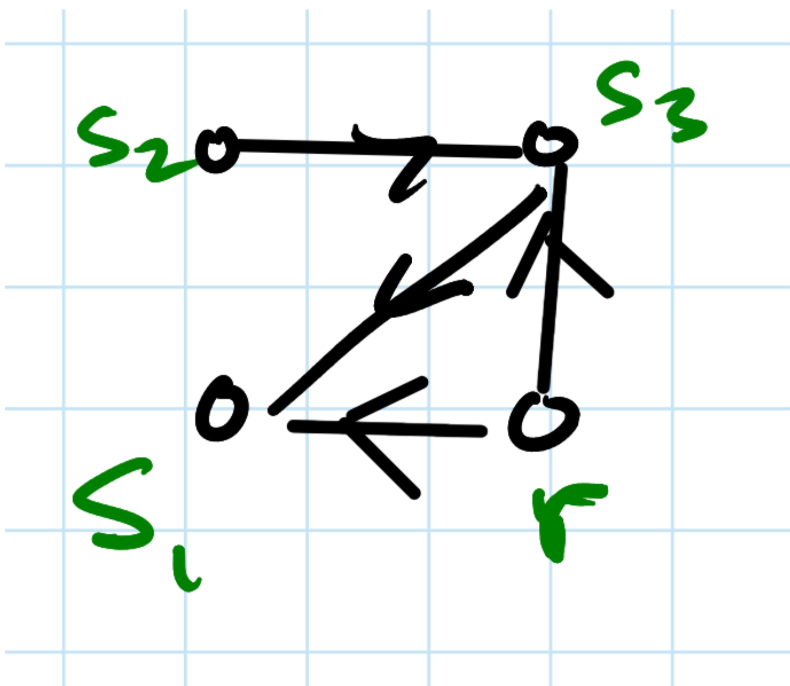
****Note: I believe that this should mean a branching.**

This makes sense because limiting the number of forests means that there is a lower bound on the number of edges, and requires that each node that isn't a root has an incoming edge.

For the four node example:

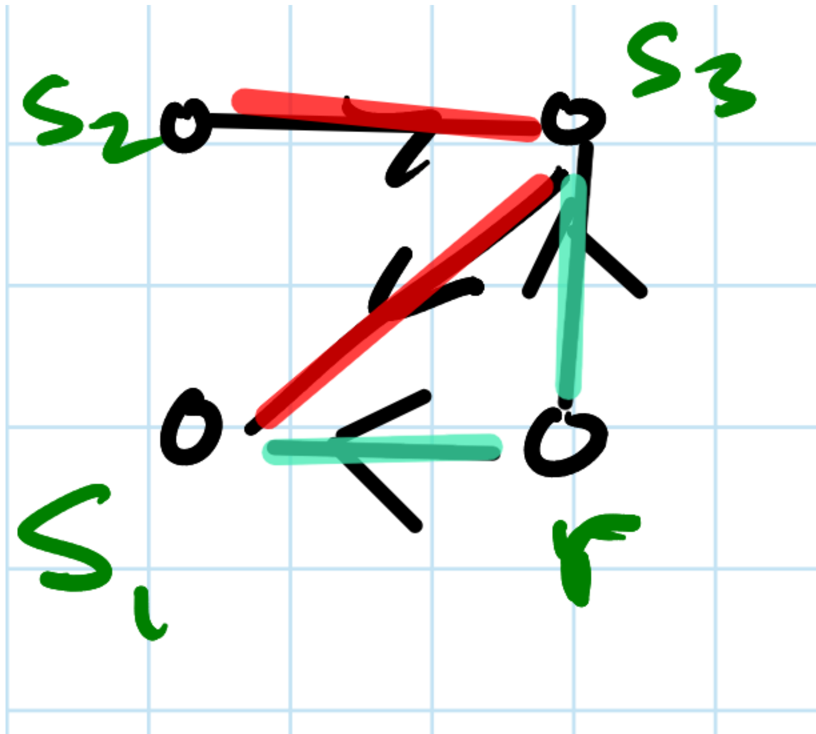
To keep things simple, let's only describe M2 intersect M1 here (ie let's consider M2 definition for only the edges without the edge S1-S2, as the matroid with that edge is not a graph in M1, and not part of the solution).

So, let's find M2 for the following:



Let's consider them by the number of edges:

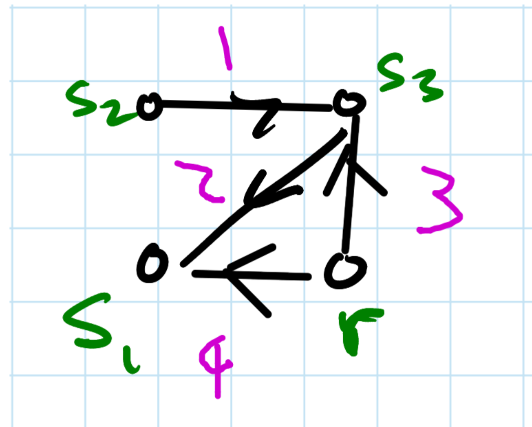
Number of edges = 4 (ie all of them).



This one belongs because we can partition the edges as above to get two different trees. Since this partitioning works, then the above belongs in the intersection. This is #4 in the listed branchings.

Number of edges = 3. Clearly removing any of s2-s3, s1-r, r-s3, and s3-s1 leaves us with a good partition. These are #9, #2, #3 and #5 respectively.

Number of edges = 2. Now it gets a bit more interesting. Let's relabel it to be a bit easier to tell:

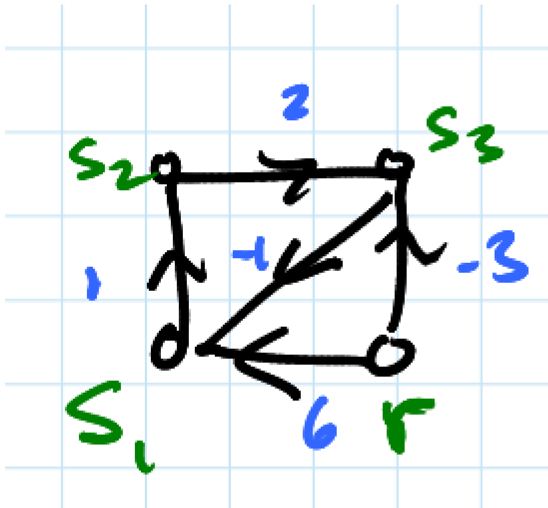


There are 6 possible combinations (4 choose 2) - {1, 2}, {1, 3}, {1, 4}, {2, 3}, {2, 4}, {3, 4}.

- {1, 2} is good (use {1, 2} as the partitioning). This corresponds with #1.
- {1, 3} is not good, as long as the definition is branching and not forest, as this would disqualify the partitioning of {1, 3}, and the only partitioning would be {{1}, {3}} which would have 3 different trees, disqualifying {1, 3}.
- {1, 4} is good (split it {{1}, {4}}) and we have two branchings only. This corresponds with #6.
- {2, 3} is good (use {2, 3}). This corresponds with #7.
- {2, 4} is bad for the same reason as {1, 3}.
- {3, 4} is good (use {3, 4}). This corresponds with #8.

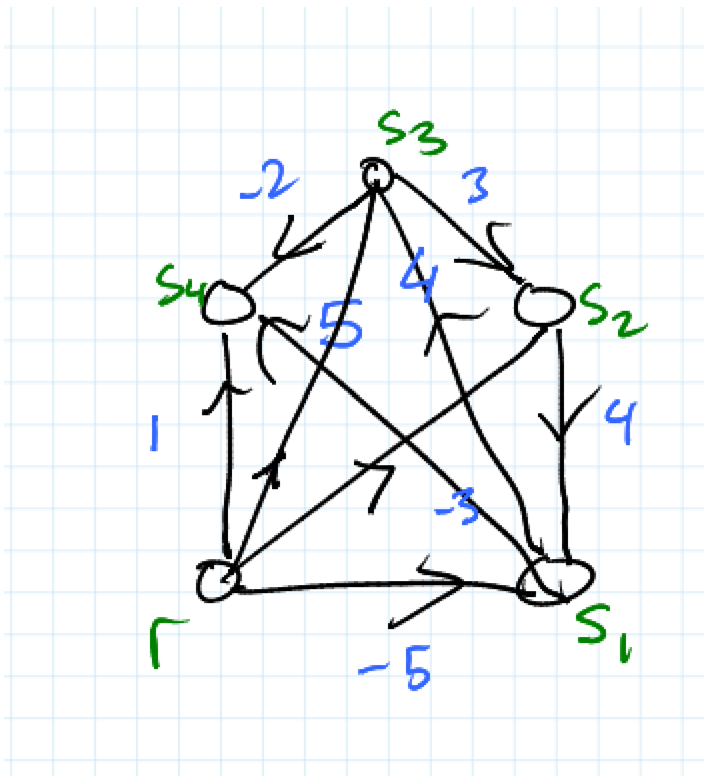
Any edge of just one leaves two other empty nodes, we have three forests no matter what (disqualifies them). And so, we've listed all of them, and this should be the only branchings we consider in terms of finding the minimum cost.

Let’s use the following weights:



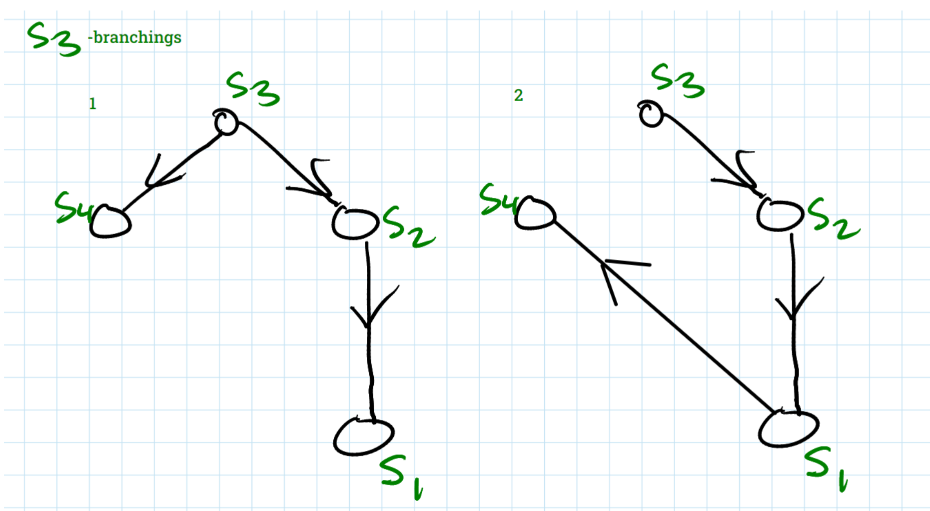
Clearly adding edges of negative weight is better, and avoiding any positive weights is good. So, we get S3-r and S3-S1 as the edges included as the optimal branching system. Let’s explore a bigger example. Let’s do a five node example with two roots (k = 2).

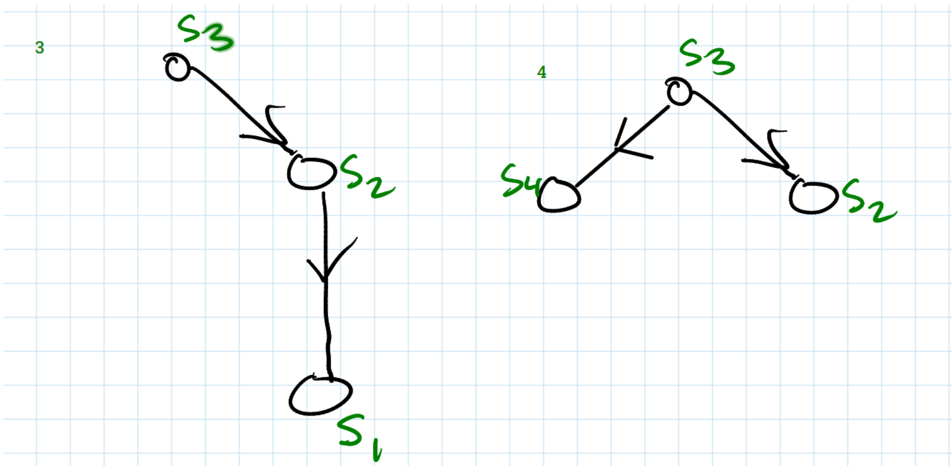
Five node example:

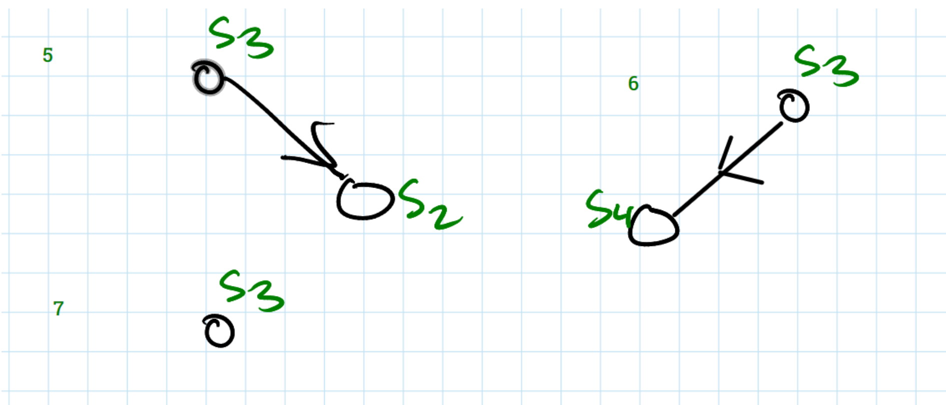


Let G be the graph above. Then, let r and S3 be the specified roots. Through brute force, let's list all the branchings possible.

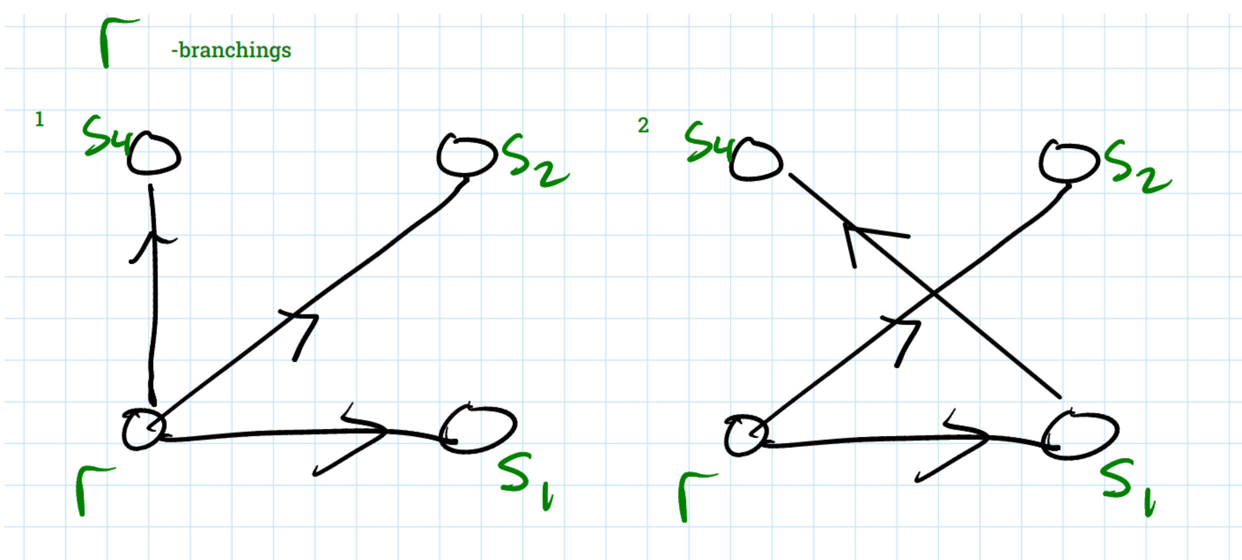
First, let's explore all the branchings rooted at S3.

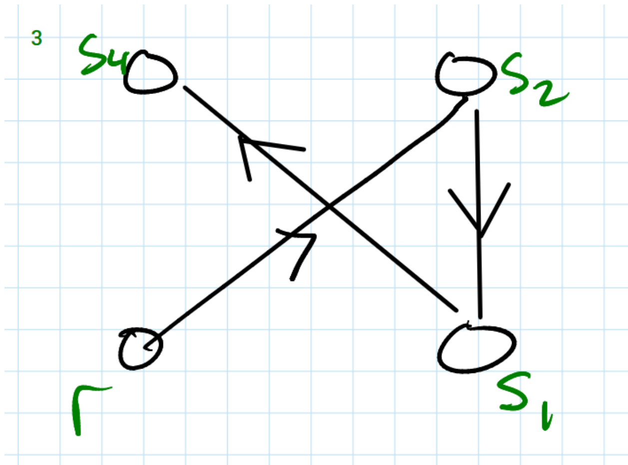


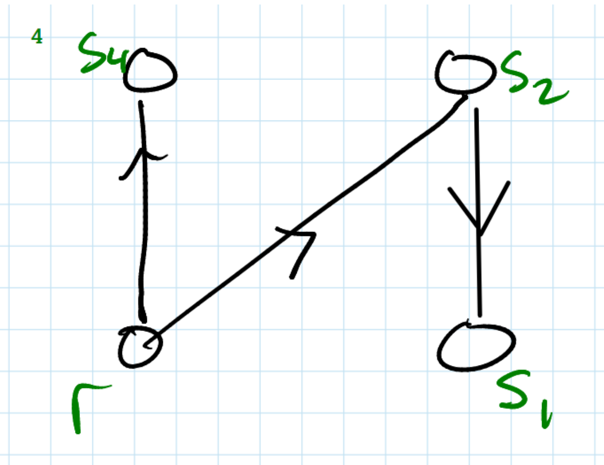


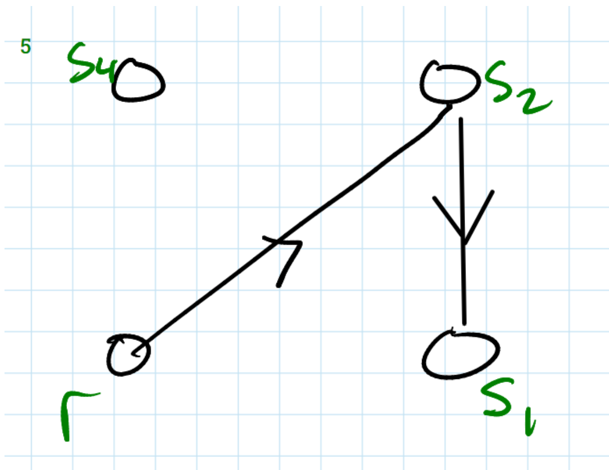


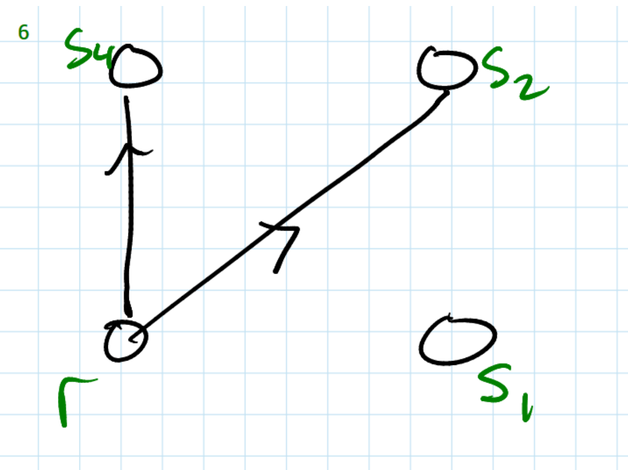
There are seven of them. Then, here are the branchings rooted at r.

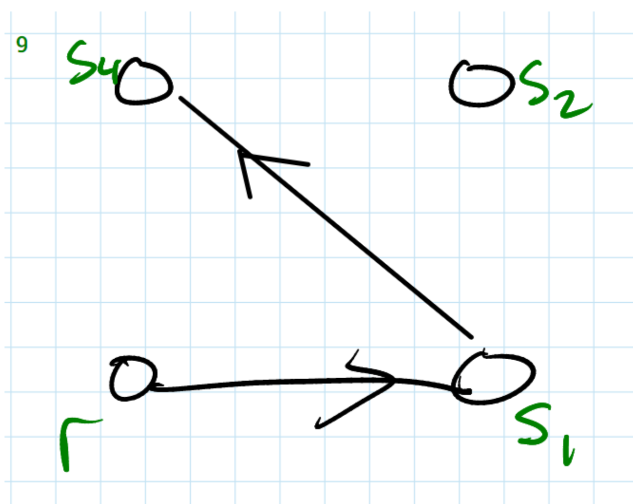


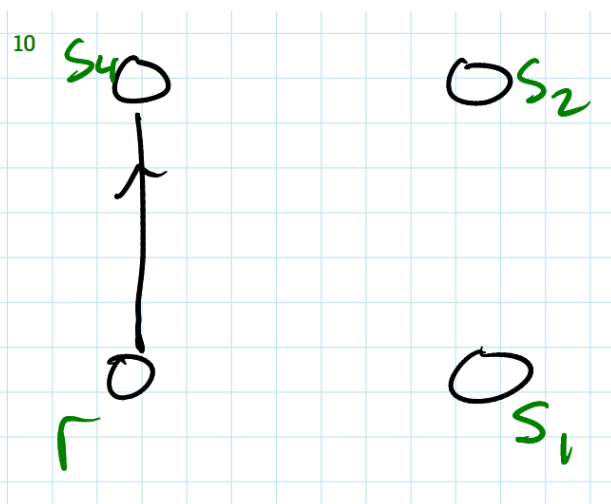

3
S4
S2
r
S1

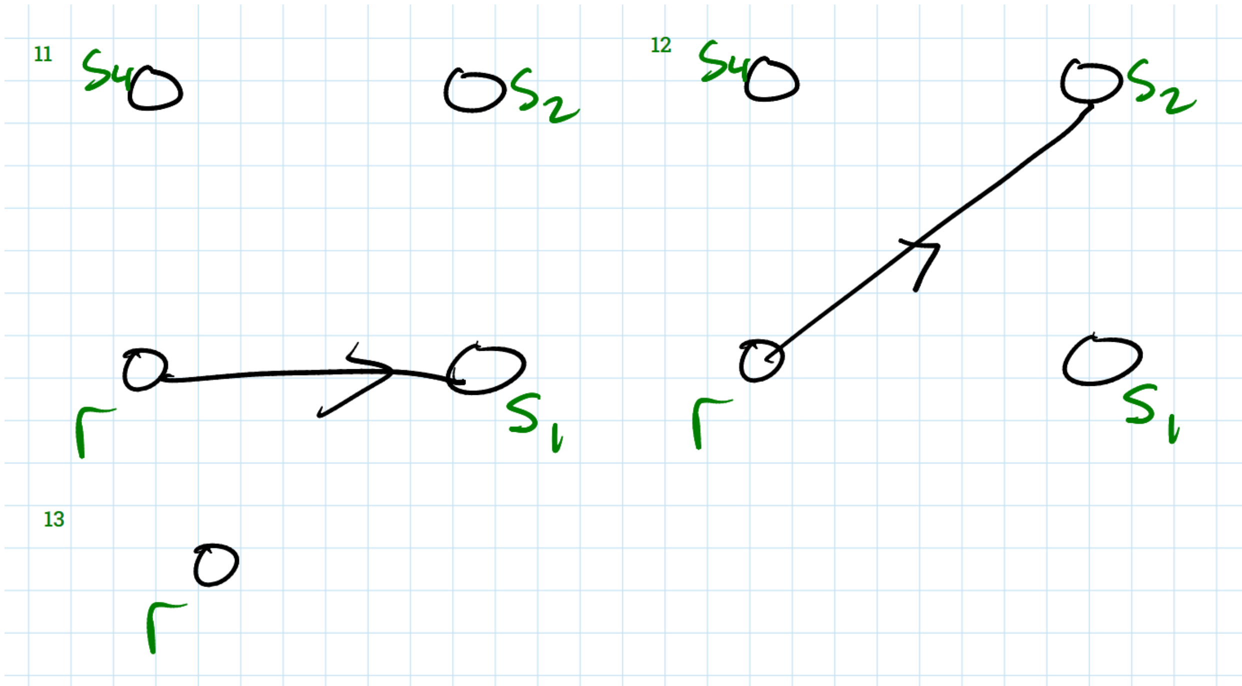

4
S4
S2
r
S1


5
S4
S2
r
S1


6
S4
S2
r
S1


9
S4
S2
r
S1


10
S4
S2
r
S1



There are thirteen possible r branchings. So what are all the possible combinations? Let's denote a branching system by a-b, ie a is an integer from 1 to 7, marking the s3-branching, and b is an integer from 1 to 13, marking the r-branching.

Possible a-1 to a-4 branchings (ie uses r = 1, 2, 3, 4 branchings):

- Since 1 to 4 from the r branchings covers every single non-root, then any combination with r3 branchings work.
- So 1-1, 2-1, …, 7-1, 1-2, 2-2, …, 7-2, …, 1-4, 2-4, …, 7-4.
- Note however, a lot of these overlap…

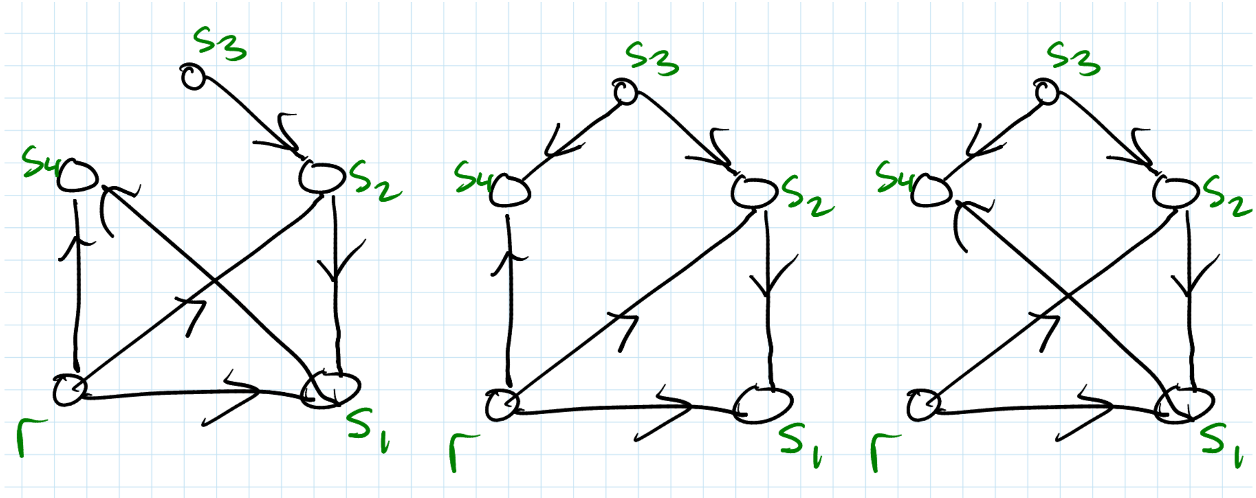
Possible a-5 branchings (ie uses r = 5):

- Anything that covers S4 would work.

I think I'm too lazy/isn't too important to list all of them out (requires looking at 91 possible branching systems!)

M1:

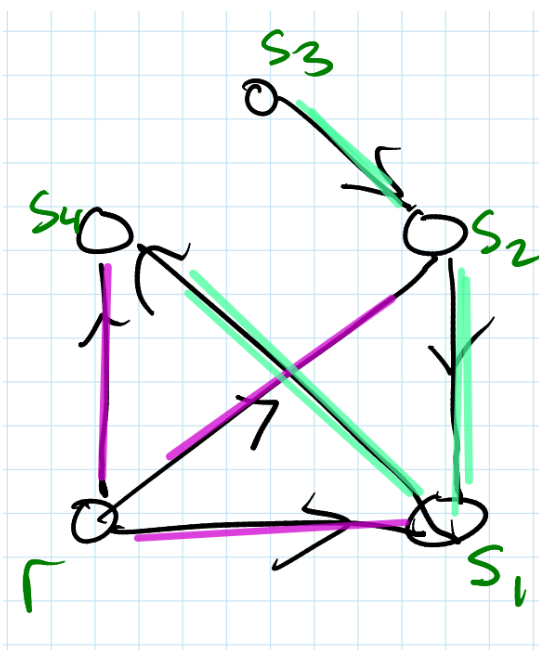
Union of the power sets of the following. Denote the leftmost as A, middle as B, and the rightmost as C.



M2:

Again, let's consider the intersection (ie apply M2 procedure to M1).

Edge count = 6, A:



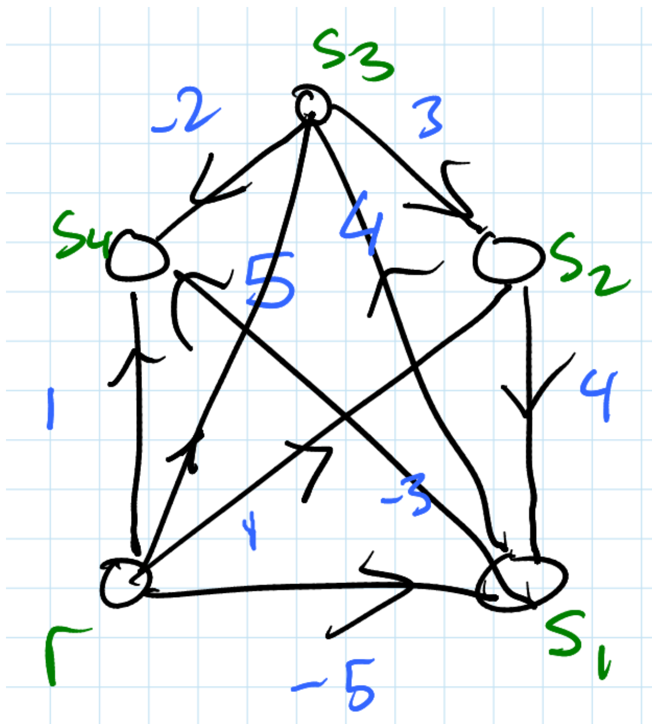
This is a branching system, this is 1-2.

Edge count = 5, A:

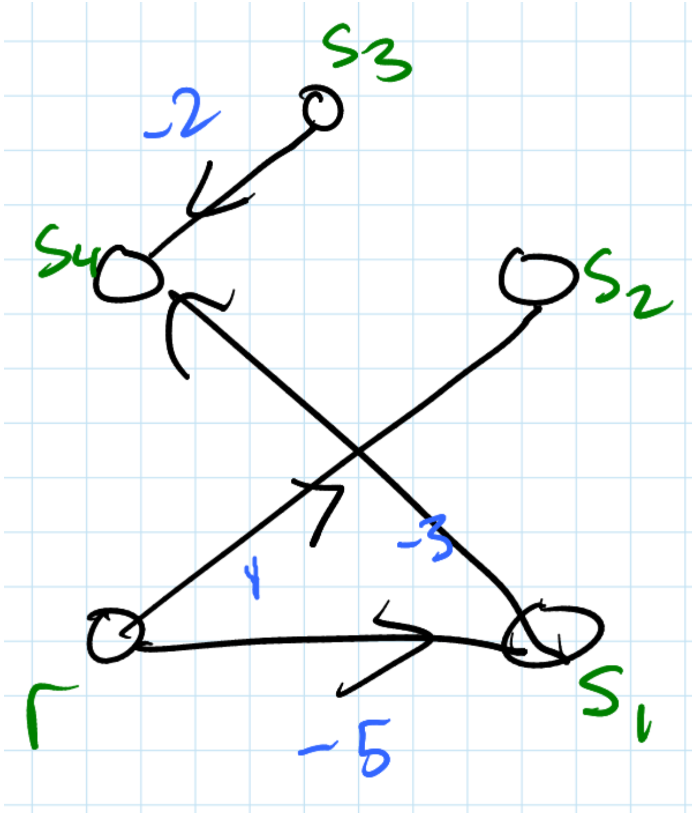
This too will take a really long time…

Here’s what we should get:

Given the weights as follows,



This should be the optimal branching system, with a total weight of -9.



This combines s3 branching of #6 with r branching of #2.