**1. Undirected graph/MST definitions:**

Please check the following files:

* undirected\_graph.v
* mst.v

Do you think this idea of connectedness and spanning\_tree is acceptable?

So every graph is effectively directed, undirected properties are built over it

I think it makes sense because every graph representation in C is directed, so no point explicitly defining undirected graphs

And every directed graph has an underlying undirected graph

However, do you want to be able to talk about mixed graphs in the future?

If so, then undirected edges must be explicitly defined, as in the wiki page for mixed graphs

Deferring to Aquinas

**2. Edgearray representation**

a) I don't understand what SDAG is in dijkstra and its importance

Because of this, I'm not sure what I'm doing by defining SWEG

UnionFind doesn't have such a thing

You’re probably right; it may be unnecessary. The reason it exists is to give flexibility about the kind of spatial representation we use. You can see (line 109 of msl\_application/DijkstraArrayGraph), SpatialDijkstraArrayGraph is defined with abstract\_data\_at (i.e., any data\_at - ish thing), and later instantialized to use VST’s data\_at and pointer\_val. (line 14 of sample\_mark/spatial\_dijkstra\_array\_graph.v). Although we’re trying to stay flexible in this way, the truth is that we only ever use VST’s representation currently. So it’s possible that many of my ostensibly generic developments are actually specific to VST. But anyway, that’s the attempt here. Separate the generic stuff from the VST-specific stuff.

b) My graph\_rep in spatial\_wedgearray\_graph.v has an issue

Basically, in the C file I malloc a SIZE array of edges

However, not all edges are used, e.g. in init\_empty\_graph there are no edges

Edges are slowly added over time (not removed in kruskal, so we don't have to deal with that)

So, in the graph\_rep I think I need something like

(tarray t\_struct\_edge N ...) \* (memory\_block ...)

To represent the existing edges, and the rest of the array

Can you look and give some advice?

I think what you have looks okay. There are many ways to do this, either using logic/hacks or using VST’s functionality. For example you could just use a *fully connected* graph, where in reality an edge weight can be some distinguished value (infinity) to model unconnectedness. There is also VST functionality (search for data\_at\_ in VST/doc/VC.pdf) to model spaces in memory that have been allocated but where we don’t know exactly what is sitting at the memory.

c) If the problem can't be dealt with elegantly, I'm thinking change it into a linked list of edges

VST has a queue.c that is very suitable, all I have to do is just add an extra (int c)

Deferring to Aquinas

**3. Proof (\*probably the most important issue\*)**

I started trying to work on init\_empty\_graph, and immediately get a complaint about invalid expression at the first forward\_call:

typecheck\_error (invalid\_expression (sizeof(Tstruct \_graph noattr))%expr)

Not sure what the problem is.

I looked at verif\_unionfind\_arr where the first thing in makeSet is the same mallocN, and they don't seem to have this problem

I can’t recreate this problem because there seems to be a bug in your graph\_rep calls. Please could you fix and push? I’ll take a look then. Note Aquinas and I are on a specific commit of VST:

…/VST$ git fetch origin

…/VST$ git checkout master

…/VST$ git merge dac41dd686728c800bc8ac66d6f71a91c905f96c

Also, I'm thinking that are too many mallocNs running around

So I tried to make a mallocN.h and matching malloc\_spec.v, and have everyone reference them

Problem is clightgen won't process a header file with only an extern, and has issues even if I move it to a mallocN.c. Consequently mallocN\_spec.v will complain about there not being a \_mallocN

But, I think I can overcome it by writing some unused function (say a freeN)

Advice on whether this is a good way to handle it

Deferring to Aquinas, perhaps Shengyi.