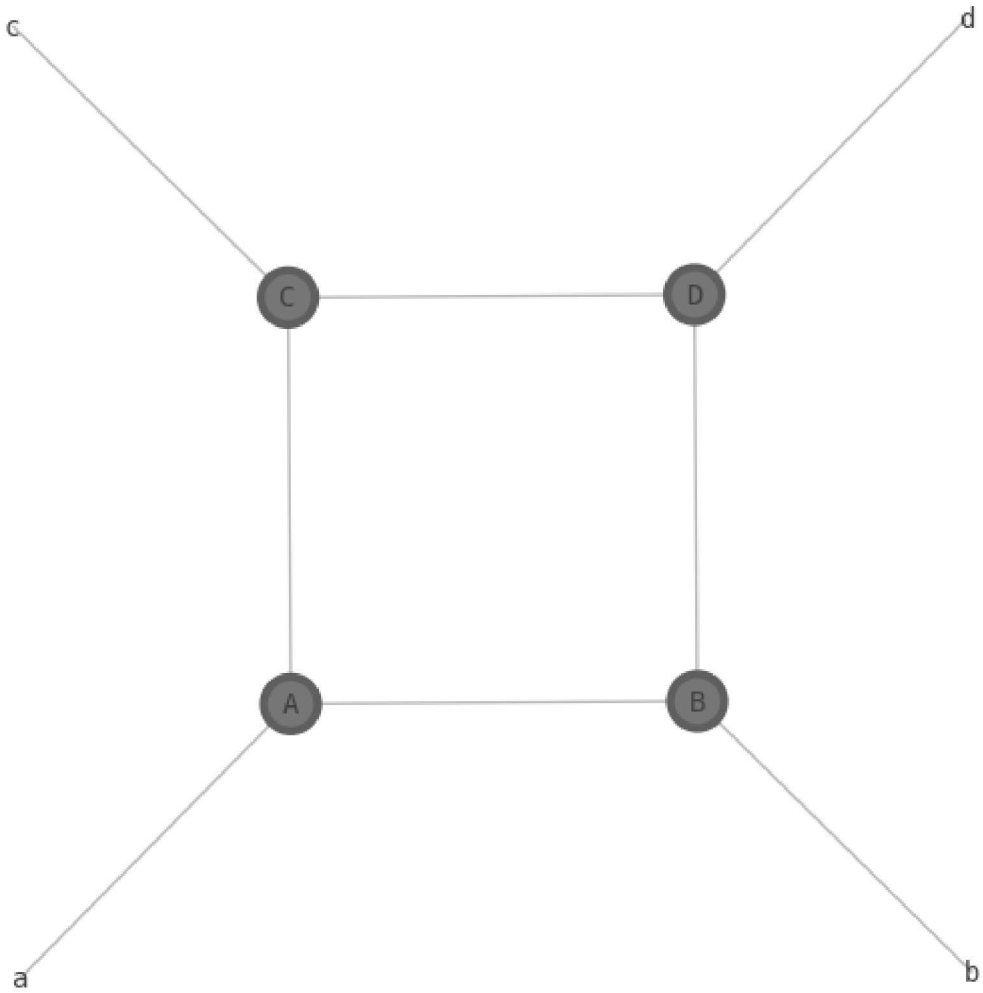
In [ ]:
```
import quimb as qu
import quimb.tensor as qtn
import quimb.experimental.tn_marginals as marginals
import numpy as np
import quimb.tensor as qtn
```

In [ ]:
```
ta = qtn.rand_tensor([2, 2, 2], inds=['a', 'w', 'x'], tags='A')
tb = qtn.rand_tensor([2, 2, 2], inds=['b', 'w', 'y'], tags='B')
tc = qtn.rand_tensor([2, 2, 2], inds=['c', 'x', 'z'], tags='C')
td = qtn.rand_tensor([2, 2, 2], inds=['d', 'y', 'z'], tags='D')
tn = (ta | tb | tc | td)
tn.draw()
```



Let's compute all the marginals in the network, i.e. let's set the output_inds to be all_inds. This throws

an error

In [ ]:
```
inds=tn.all_inds()
posteriors=marginals.compute_all_marginals_via_slicing(tn,output_inds=inds)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[30], line 2
      1 inds=tn.all_inds()
----> 2 posteriors=marginals.compute_all_marginals_via_slicing(tn,output_inds=inds)

File c:\Users\apiedraf\VS_projects\mbdlyb\.conda\Lib\site-packages\quimb\experimenta
l\tn_marginals.py:39, in compute_all_marginals_via_slicing(tn, output_inds, optimiz
e, **contract_kwargs)
     34 wv = [0.0 for i in range(tree_v.size_dict[symbol])]
     36 for s in range(tree_v.nslices):
     37
     38     # contract the slice
---> 39     p, exponent = tree_v.contract_slice(
     40         arrays, s, **contract_kwargs, strip_exponent=True
     41     )
     43     if overall_exponent is None:
     44         # set overall exponent from first slice...
     45         overall_exponent = exponent

File c:\Users\apiedraf\VS_projects\mbdlyb\.conda\Lib\site-packages\cotengra\core.py:
2675, in ContractionTree.contract_slice(self, arrays, i, **kwargs)
   2673 def contract_slice(self, arrays, i, **kwargs):
   2674     """Get slices ``i`` of ``arrays`` and then contract them."""
-> 2675     return self.contract_core(self.slice_arrays(arrays, i), **kwargs)

File c:\Users\apiedraf\VS_projects\mbdlyb\.conda\Lib\site-packages\cotengra\core.py:
2619, in ContractionTree.contract_core(self, arrays, order, prefer_einsum, strip_exp
onent, check_zero, backend, implementation, autojit, progbar)
   2589 """Contract ``arrays`` with this tree. The order of the axes and
   2590 output is assumed to be that of ``tree.inputs`` and ``tree.output``,
   2591 but with sliced indices removed. This functon contracts the core tree
   (...)
   2610     Show progress through the contraction.
   2611 """
   2612 fn = self.get_contractor(
   2613     order=order,
   2614     prefer_einsum=prefer_einsum,
   (...)
   2617     autojit=autojit,
   2618 )
-> 2619 result = fn(
   2620     *arrays,
   2621     check_zero=check_zero,
   2622     backend=backend,
   2623     progbar=progbar,
   2624 )
   2626 # handle exponent outside of potential jit
   2627 if isinstance(strip_exponent, dict):

File c:\Users\apiedraf\VS_projects\mbdlyb\.conda\Lib\site-packages\cotengra\contrac
t.py:712, in do_contraction(contractions, strip_exponent, check_zero, implementatio
n, backend, progbar, *arrays)
    709 for p, l, r, tdot, arg, perm in contractions:
    710     if (l is None) and (r is None):
    711         # single term simplification, perform inplace with einsum
```

```
--> 712         temps[p] = _einsum(arg, temps[p])
    713         continue
    715     # get input arrays for this contraction

File c:\Users\apiedraf\VS_projects\mbdlyb\.conda\Lib\site-packages\cotengra\contrac
t.py:424, in einsum(eq, a, b, backend)
    403 """Perform arbitrary single and pairwise einsums using only `matmul`,
    404 `transpose`, `reshape` and `sum`.  The logic for each is cached based on
    405 the equation and array shape, and each step is only performed if necessary.
   (...)
    421 array_like
    422 """
    423 if b is None:
--> 424     return _einsum_single(eq, a, backend=backend)
    426 (
    427     eq_a,
    428     eq_b,
   (...)
    433     pure_multiplication,
    434 ) = _parse_eq_to_batch_matmul(eq, shape(a), shape(b))
    436 return _do_contraction_via_bmm(
    437     a,
    438     b,
   (...)
    446     backend,
    447 )

File c:\Users\apiedraf\VS_projects\mbdlyb\.conda\Lib\site-packages\cotengra\contrac
t.py:335, in _einsum_single(eq, x, backend)
    329 """Einsum on a single tensor, via three steps: diagonal selection
    330 (via advanced indexing), axes summations, transposition. The logic for each
    331 is cached based on the equation and array shape, and each step is only
    332 performed if necessary.
    333 """
    334 try:
--> 335     return do("einsum", eq, x, like=backend)
    336 except ImportError:
    337     pass

File c:\Users\apiedraf\VS_projects\mbdlyb\.conda\Lib\site-packages\autoray\autoray.p
y:79, in do(fn, like, *args, **kwargs)
     30 """Do function named ``fn`` on ``(*args, **kwargs)``, peforming single
     31 dispatch to retrieve ``fn`` based on whichever library defines the class of
     32 the ``args[0]``, or the ``like`` keyword argument if specified.
   (...)
     76     <tf.Tensor: id=91, shape=(3, 3), dtype=float32>
     77 """
     78 backend = choose_backend(fn, *args, like=like, **kwargs)
---> 79 return get_lib_fn(backend, fn)(*args, **kwargs)

File <__array_function__ internals>:200, in einsum(*args, **kwargs)

File c:\Users\apiedraf\VS_projects\mbdlyb\.conda\Lib\site-packages\numpy\core\einsum
func.py:1371, in einsum(out, optimize, *operands, **kwargs)
   1369     if specified_out:
   1370         kwargs['out'] = out
```

```
-> 1371     return c_einsum(*operands, **kwargs)
   1373 # Check the kwargs to avoid a more cryptic error later, without having to
   1374 # repeat default values here
   1375 valid_einsum_kwargs = ['dtype', 'order', 'casting']

ValueError: einstein sum subscripts string contains too many subscripts for operand
0
```

If we limit ourselves to external indices, it seems like things work again, although the numbers returned don't make sense

In [ ]:
```
inds=['a','b','c','d']
posteriors=marginals.compute_all_marginals_via_slicing(tn,output_inds=inds)
posteriors
```

Out[ ]:
```
{'a': ([1.0, 1.0], 1.0394829936365781),
 'b': ([1.0, 1.0], 1.0394829936365781),
 'c': ([1.0, 1.0], 1.0394829936365781),
 'd': ([1.0, 1.0], 1.0394829936365781)}
```

In [ ]:
```
inds=['a']
marginals2=marginals.compute_all_marginals_via_slicing(tn,output_inds=inds)
marginals2
```

Out[ ]:
```
{'a': ([1.0, 1.0], 1.0394829936365781)}
```

However, computing the marginal of an internal index does not seem to work at all

In [ ]:
```
inds=['w']
marginals3=marginals.compute_all_marginals_via_slicing(tn,output_inds=inds)
marginals3
```

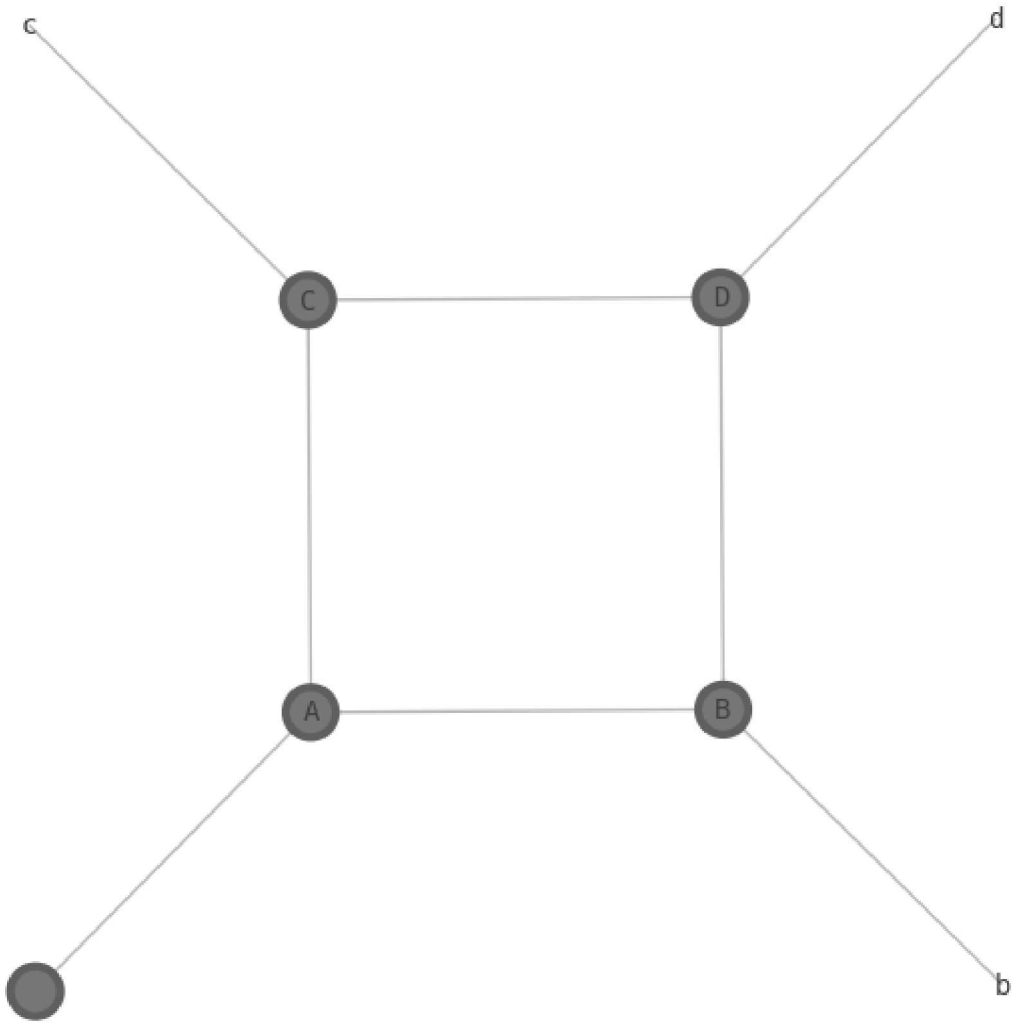
```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[16], line 2
      1 inds=['w']
----> 2 posteriors3=marginals.compute_all_marginals_via_slicing(tn,output_inds=inds)
      3 posteriors3

File c:\Users\apiedraf\VS_projects\mbdlyb\.conda\Lib\site-packages\quimb\experimenta
l\tn_marginals.py:39, in compute_all_marginals_via_slicing(tn, output_inds, optimiz
e, **contract_kwargs)
     34 wv = [0.0 for i in range(tree_v.size_dict[symbol])]
     36 for s in range(tree_v.nslices):
     37
     38     # contract the slice
---> 39     p, exponent = tree_v.contract_slice(
     40         arrays, s, **contract_kwargs, strip_exponent=True
     41     )
     43     if overall_exponent is None:
     44         # set overall exponent from first slice...
     45         overall_exponent = exponent

File c:\Users\apiedraf\VS_projects\mbdlyb\.conda\Lib\site-packages\cotengra\core.py:
2675, in ContractionTree.contract_slice(self, arrays, i, **kwargs)
   2673 def contract_slice(self, arrays, i, **kwargs):
   2674     """Get slices ``i`` of ``arrays`` and then contract them."""
-> 2675     return self.contract_core(self.slice_arrays(arrays, i), **kwargs)

File c:\Users\apiedraf\VS_projects\mbdlyb\.conda\Lib\site-packages\cotengra\core.py:
2619, in ContractionTree.contract_core(self, arrays, order, prefer_einsum, strip_exp
onent, check_zero, backend, implementation, autojit, progbar)
   2589 """Contract ``arrays`` with this tree. The order of the axes and
   2590 output is assumed to be that of ``tree.inputs`` and ``tree.output``,
   2591 but with sliced indices removed. This functon contracts the core tree
   (...)
   2610     Show progress through the contraction.
   2611 """
   2612 fn = self.get_contractor(
   2613     order=order,
   2614     prefer_einsum=prefer_einsum,
   (...)
   2617     autojit=autojit,
   2618 )
-> 2619 result = fn(
   2620     *arrays,
   2621     check_zero=check_zero,
   2622     backend=backend,
   2623     progbar=progbar,
   2624 )
   2626 # handle exponent outside of potential jit
   2627 if isinstance(strip_exponent, dict):

File c:\Users\apiedraf\VS_projects\mbdlyb\.conda\Lib\site-packages\cotengra\contrac
t.py:712, in do_contraction(contractions, strip_exponent, check_zero, implementatio
n, backend, progbar, *arrays)
    709 for p, l, r, tdot, arg, perm in contractions:
    710     if (l is None) and (r is None):
```

```
    711         # single term simplification, perform inplace with einsum
--> 712         temps[p] = _einsum(arg, temps[p])
    713         continue
    715     # get input arrays for this contraction

File c:\Users\apiedraf\VS_projects\mbdlyb\.conda\Lib\site-packages\cotengra\contract.py:424, in einsum(eq, a, b, backend)
    403 """Perform arbitrary single and pairwise einsums using only `matmul`,
    404 `transpose`, `reshape` and `sum`.  The logic for each is cached based on
    405 the equation and array shape, and each step is only performed if necessary.
   (...)
    421 array_like
    422 """
    423 if b is None:
--> 424     return _einsum_single(eq, a, backend=backend)
    426 (
    427     eq_a,
    428     eq_b,
   (...)
    433     pure_multiplication,
    434 ) = _parse_eq_to_batch_matmul(eq, shape(a), shape(b))
    436 return _do_contraction_via_bmm(
    437     a,
    438     b,
   (...)
    446     backend,
    447 )

File c:\Users\apiedraf\VS_projects\mbdlyb\.conda\Lib\site-packages\cotengra\contract.py:335, in _einsum_single(eq, x, backend)
    329 """Einsum on a single tensor, via three steps: diagonal selection
    330 (via advanced indexing), axes summations, transposition. The logic for each
    331 is cached based on the equation and array shape, and each step is only
    332 performed if necessary.
    333 """
    334 try:
--> 335     return do("einsum", eq, x, like=backend)
    336 except ImportError:
    337     pass

File c:\Users\apiedraf\VS_projects\mbdlyb\.conda\Lib\site-packages\autoray\autoray.py:79, in do(fn, like, *args, **kwargs)
     30 """Do function named ``fn`` on ``(*args, **kwargs)``, peforming single
     31 dispatch to retrieve ``fn`` based on whichever library defines the class of
     32 the ``args[0]``, or the ``like`` keyword argument if specified.
   (...)
     76     <tf.Tensor: id=91, shape=(3, 3), dtype=float32>
     77 """
     78 backend = choose_backend(fn, *args, like=like, **kwargs)
---> 79 return get_lib_fn(backend, fn)(*args, **kwargs)

File <__array_function__ internals>:200, in einsum(*args, **kwargs)

File c:\Users\apiedraf\VS_projects\mbdlyb\.conda\Lib\site-packages\numpy\core\einsumfunc.py:1371, in einsum(out, optimize, *operands, **kwargs)
   1369     if specified_out:
```

```
   1370         kwargs['out'] = out
-> 1371     return c_einsum(*operands, **kwargs)
   1373 # Check the kwargs to avoid a more cryptic error later, without having to
   1374 # repeat default values here
   1375 valid_einsum_kwargs = ['dtype', 'order', 'casting']

ValueError: einstein sum subscripts string contains too many subscripts for operand
0
```

Or doesn't it? Let's make a slight change to the network and make 'a' an internal index

In [ ]:
```
prior_a = qtn.Tensor(np.array([1,1]),inds=['a'])
tn2 = tn = (ta | tb | tc | td| prior_a)
tn2.draw()
```



In [ ]:
```
inds=['a']
marginals4=marginals.compute_all_marginals_via_slicing(tn2,output_inds=inds)
```

```
marginals4
```

Out[ ]:
```
{'a': ([1.0, 0.04998480401290142], 0.832398465393601)}
```

So now it seems to compute well, eventhough I see no difference between the role of 'a' now and 'w' before

Let us now pass 'a','b','c','d' as output_inds. The output has a very strange form now.

In [ ]:
```
inds=['a','b','c','d']
marginals5=marginals.compute_all_marginals_via_slicing(tn2,output_inds=inds)
marginals5
```

Out[ ]:
```
{'a': ([1.0, 0.04998480401290142], 0.832398465393601),
 'b': ([array([[-0.08943496, -0.39483732],
         [ 0.2265109 ,  1.        ]]),
   array([[-0.08943496, -0.39483732],
         [ 0.2265109 ,  1.        ]])],
  1.1059195498920746),
 'c': ([array([[-0.08943496, -0.39483732],
         [ 0.2265109 ,  1.        ]]),
   array([[-0.08943496, -0.39483732],
         [ 0.2265109 ,  1.        ]])],
  1.1059195498920746),
 'd': ([array([[-0.08943496, -0.39483732],
         [ 0.2265109 ,  1.        ]]),
   array([[-0.08943496, -0.39483732],
         [ 0.2265109 ,  1.        ]])],
  1.1059195498920746)}
```

Finally, let's compute marginals through a naif implementation

We can obtain these marginals through naively contracting all but one edge in the network, and iterating over the left-out edge.

In [ ]:
```
def get_posteriors(Network: qtn.TensorNetwork,targets: list[str]=None):
    result={}
    inds=targets if targets!= None else Network.ind_map
    for ind in inds:
        temp=Network.contract(output_inds=[ind]).data
        val=temp/np.sum(temp)
        result[ind]= val
    return result
```

In [ ]:
```
marginals6=get_posteriors(tn2,targets=tn2.all_inds())
marginals6
```

Out[ ]:
```
{'a': array([0.95239474, 0.04760526]),
 'w': array([0.95487029, 0.04512971]),
 'x': array([0.66756425, 0.33243575]),
 'b': array([0.76423412, 0.23576588]),
 'y': array([0.0461156, 0.9538844]),
 'c': array([0.15932099, 0.84067901]),
 'z': array([0.91518685, 0.08481315]),
 'd': array([0.08442477, 0.91557523])}
```