3.

1.alg

First, for e in S, we treat them as not exist in G(but their data is kept). So it's now G(V,E)

Run DFS

- If there are more than 2 connected component, return directly
- If there are 2 connected component, we say it's situation A
- If it's connected, we say it's situation B

Run prim(If it's A, we run prim in 2 connected components)

If B:(in this situation, add other edge must form a cycle)

```
for e in S:
    treat e as existed
    use find-union to find the cycle:
        denote edge e_1 in cycle with max weight e_1
        if c(e_1)>c(e):
            label e and assign the new T's weight to e
        mark e as not exist and visited
return the minimum T
```

If A:

```
for e in S:
    if both vertices belongs to 2 connected component:
        add e to S_1
for e in S_1:
    run DFS to check whether it's connected
    label e and assign the new T's weight to e and mark as not exist and visited
return the minimum T
```

2.Correctness

The situation that we have a tree in G with at most one edge belongs to S is A and B.

For A, we only need to go through edge that connects the 2 component and choose the minimum tree

For B, we form a cycle every time we add a edge in S(Also there is only one. Otherwise, we can form a big cycle from 2 small cycle after deleting e) . We just need to check out whether we need to delete one edge in cycle to make it smaller.

3.Runtime

DFS is O(|V|+|E|)

Prim is O(|E|log|V|)

In B, we do find-union in each loop so it's O(|S|log|V|)

In A, we use a loop to form S_1. We can label each vertex in a array to reveal which component they are in which is done by union find(log|V|) and the the check is O(1). So it's O(|S|log|V|)The loop in S_1 is O(1) for each. So it's O(|S|log|V|+|S_1|)) = O(|S|log|V|)

Total runtime = O(|E|log|V|+|S|log|V|)

4.

1.Alg

define each state with (i,j)(n*n vertices). If include edge e in (i,j)-->(input(i,k),input(j,k)) in the graph.

first, run dfs to find all SCCs in the graph. For each SCC, compress them to one vertex and form a new graph(SCC graph). To form edge's in the new graph, randomly pick 2 vertex in each SCC and run dfs with one vertex as start and the other as end. In the new graph, we reverse every edge and try to find a mother vertex(from this vertex, we can go to every other vertex) based on Kosaraju's algorithm discussed in class.(the mother vertex is a sink in original graph,If there exist mother vertex (or vertices), then one of the mother vertices is the last finished vertex in DFS.). If we can't find one,return false. If we find one, go to that SCC and check whether it has (i,i). If it has, return true. Otherwise, return false.

2.Correctness

the edge in graph represents each portal. Notice that (i,i) can only goes to (j,j). If we wants all PNP end at the same igloo, then there should be a (k,k) in the graph that every vertex can goes to. Thus, if (k,k) exist, it should be in the SSCC. So we first use SCC algorithm to divide each group. In the SCC graph, we reverse edge to find their mother vertex. The mother vertex is the SSCC. Then we only need to check whether it has a (k,k) to trap all the PNP. Because if the PNP goes into SSCC. We can just ignore them because they can't go out. After move all PNP into it, we can use (k,k) to trap all of them since it's a SCC.

3.Runtime

let n be number of igloos and k be number of portals

Construct graph. 2 loop for n with 1 loop for k O(k*n^2)

find SCC. since it's dfs. it's O(|V|+|E|) = O(n^2+kn^2)

form SCC graph. Randomly pick 2 from different SCC, worst case, each is a SCC O(n^2)

find mother vertex. O(|V|+|E|) = O(n^2+kn^2)

find (i,i) in that SSCC. O(n^2)

Total runtime = O(|V|+|E|) = O(k*n^2)