CS 170 Homework 4

Due **2021-09-27, at 10:00 pm**

1 Study Group

List the names and SIDs of the members in your study group. If you have no collaborators, you must explicitly write none.

In addition, we would like to share correct student solutions that are well-written with the class after each homework. Are you okay with your correct solutions being used for this purpose? Answer "Yes", "Yes but anonymously", or "No"

2 Running Errands

You need to run a set of k errands in Berkeley. Berkeley is represented as a directed weighted graph G, where each vertex v is a location in Berkeley, and there is an edge (u, v) with weight w_{uv} if it takes w_{uv} minutes to go from u to v. The errands must be completed in order, we'll assume the ith errand can be completed immediately upon visiting any vertex in the set S_i (for example, if you need to buy snacks, you could do it at any grocery store). Your home in Berkeley is the vertex h.

Given G, h, and all S_i as input, given an efficient algorithm that computes the time needed to complete all the errands starting at h. That is, find the shortest path in G that starts at h and passes through a vertex in S_1, then a vertex in S_2, then in S_3, etc.

Give a 3-part solution.

3 MST Variant

Give an undirected graph $G = (V, E \cup S)$ with edge weight $c(e)$. Note that S is disjoint with E. Design an algorithm to find a minimum one among all spanning trees having at most one edge from S and others from E.

Input: A graph $G = (V, E \cup S)$, and a cost function $c(e)$ defined for every $e \in E \cup S$.

Output: A tree $T = (V, E')$ such that T is connected (there is a path in T between any two vertices in V), $E' \subseteq E \cup S$, $\sum_{e \in E'} c(e)$ is minimized, and $|E' \cap S| \leq 1$.

Give a 3-part solution.

4 Blackout in the Igloos

It's the year 3077, exactly 1000 years after Cyberpunk happened. PNPenguins live in igloos that are equipped with one-directional portals, allowing them to instantly travel from one igloo to another. It is polar night now, so lights are necessary for PNPenguins to navigate inside their igloos – without light, igloos are completely dark inside. There are n igloos in total, and in each igloo, there are k one-directional portals numbered $\{1 \ldots k\}$. Each portal can teleport PNPenguins to a certain igloo (including the one that this portal is located).

1.

3037534676 Shenghan Zheng

2. ① Alg

first make k copies of the graph

namely $G_1, \cdots, G_k$

link vertices of S_1 in G to their copies in G_1

and make their weight 0

$i := 2$ to k

link vertices of S_i in G_{i-1} to their copies in G_i

Label their weight to 0

Let them form a graph G'

Run dijkstra in G'. Pick the shortest path

from r in G to vertices of S_k in G_k. If all

distances of them are ∞, then the path doesn't exist

(To return the path, store each vertex's previous vertex

during dijkstra)

② Correctness

denote $G = G_0$

From G_i to reach to G_{i+1}. it must pass the vertices in S_{i+1}. Because we only connect vertices of S_{i+1} in G_i to their copies in G_{i+1} $(i=0, \dots, k-1)$

Thus, the minimum path from r in G to vertex of S_k in G_k is the desired path.

If it's ∞, then the path doesn't exist

③ Runtime

number of all edges $= (k+1)\,|E| + \sum_{i=1}^{k} |S_i|$

number of vertices $= (k+1)\,|V|$

$\Rightarrow$ runtime $= O\left(k|E| + \sum_{i=1}^{k} |S_i| + k|V| \log k|V| \right)$

using a fibonacci heap

3.

① Alg

input: described in stem

output: described in stem

first if $e \in S$, we treat it as not existed

run DFS, If there are more than 2 connected component, return directly.

run prim in $G(V,E)$. For situation that it returns a T we say it's situation A. Otherwise, we say it's situation B (2 connected component)

for $e \in S$, treat e as existed

run find-union. Notice that if it's A then add e must gives a cycle

- If A

① If the max weight edge $e' \in E$ in the cycle is bigger than $C(e)$, label that edge (e) and assign the total weight of new T to it. Then treat e as not existed

- If B

check if the 2 vertices of e belongs to 2 connected component $\xrightarrow{\text{True}}$ the graph is connected

$\xrightarrow{\text{false}}$ graph not connected

1) If it's connected, label edge e and assign total weight of new T to it. Then treat e as not existed

2) If it's not connected, treat e as not existed and move to next edge in S

② Correctness

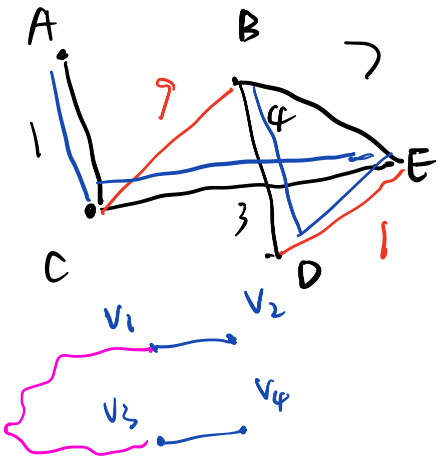
③ Runtime

prim takes $O(|E|\log|V|)$

DFS takes $O(|V|+|E|)$

For the loop 1) If it's A, we run union-find every time → $O(|S|\log|V|)$

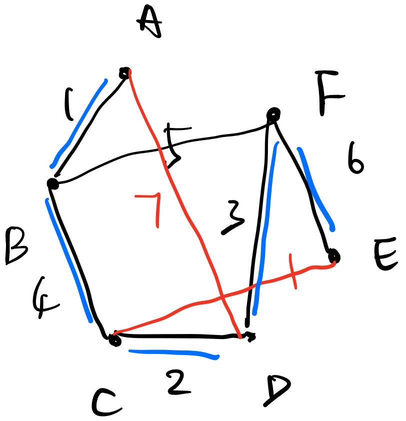
2) If it's B



first run prim

Start with lowest weight

minimum cost



Recently, PG&E (Penguin Gaming and Electric Company - yes, gaming is considered a utility in 3077) issued a notice indicating power could be shut off at any time. PNPenguins want to buy an emergency power generator for one igloo, and in the case of a power shut off, all PNPenguins need to rendezvous there. PNPenguins are terrible at memorizing which igloo they are in, but they all have a very good sense of direction. When the power is shut off at an igloo, they still can choose any portal and travel through it to another igloo.

In the event of a power shut off, PNPenguins need a way to know how to get to the igloo with the power generator. Thus, there has to be a fixed sequence of numbers $p_1, ..., p_l$ such that starting from any igloo, if a PNPenguin goes into portals p_1, ..., p_l, it ends up at the igloo with the power generator.

PNPenguins are wondering, if there exists any fixed sequence of portal numbers $p_1, ..., p_l$, such that regardless of the igloo they are currently in, after going through these portals, they end up in the same igloo? They only need to know if such a sequence of portals exist without knowing the sequence itself.

More formally, there are k functions $f_1, ..., f_k$ and each function maps each igloo to an igloo (possible the same). Your goal is to verify if there exists a sequence of numbers $p_1, ..., p_l$ such that $f_{p_1} \circ f_{p_2} \circ \cdots \circ f_{p_{l-1}} \circ f_{p_l}$ outputs the same igloo regardless of the igloo a PNPenguin starts from, i.e.,

$$\forall x, y \in \{1 \dots n\} f_{p_1} \circ f_{p_2} \circ \cdots \circ f_{p_{l-1}} \circ f_{p_l}(x) = f_{p_1} \circ f_{p_2} \circ \cdots \circ f_{p_{l-1}} \circ f_{p_l}(y)$$

$\circ$ is function composition:

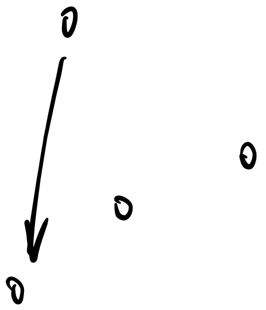
$$f_{p_1} \circ f_{p_2} \circ \cdots \circ f_{p_{l-1}} \circ f_{p_l}(x) = f_{p_1}(f_{p_2}(\dots(f_{p_{l-1}}(f_{p_l}(x)))))$$

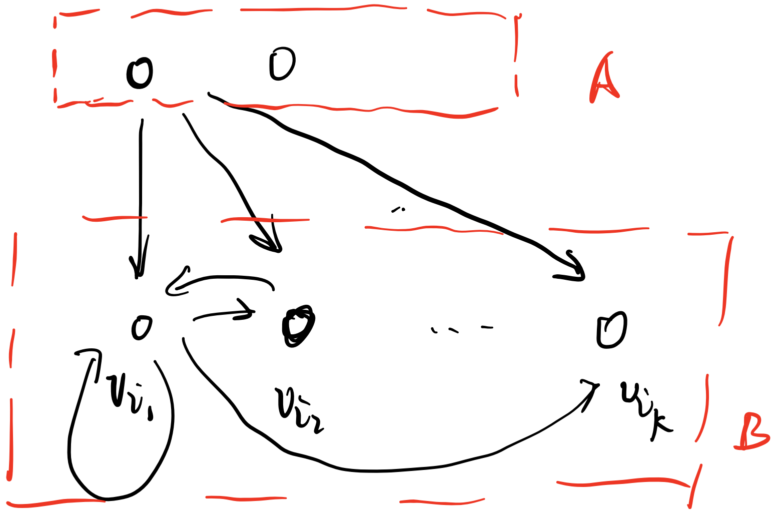
(a) Provide a 3-part solution for the described problem.

(b) Go to the online contest system at https://hellfire.ocf.berkeley.edu/, access the HW4 contest and provide a program to solve this problem.

Hint First, think about just two igloos. Can you find an algorithm that will find if there is a sequence of doors that will map these two igloos to the same igloo? Then, try to generalize this to all igloos by "merging" together pairs of igloos. Based on this, try to find an "if and only if" condition for the existence of a sequence of doors that will map all igloos to the same igloo. Can you develop an algorithm to verify if this "if and only if" condition holds?

Note that there are several ways to solve this problem with different time complexities, ranging from a very slow to a very fast solution. This is a hard problem and we do not expect everyone to get to the best possible solution. Because of this, we will be awarding partial credit for slower solution, even for a solution that is exponential in the input size. Try to do your best. We will be providing Office Hours support for this question, but note that we will emphasize conceptual help and we do not guarantee that a staff member will be able to debug your code.

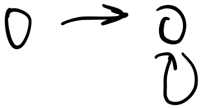


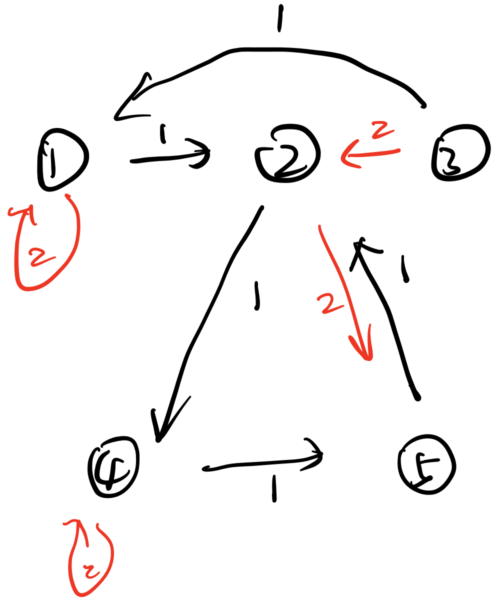


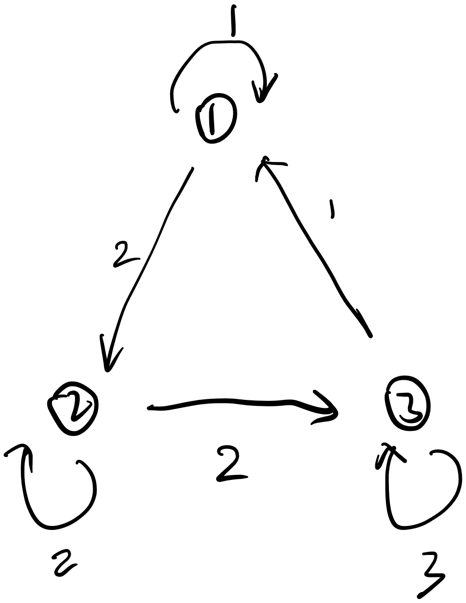
K_k

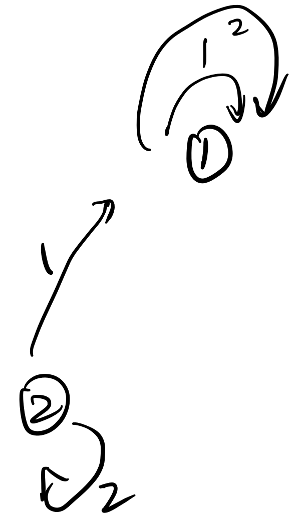
destination must

in B

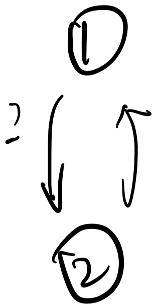






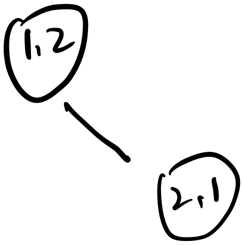


1 2
2 1



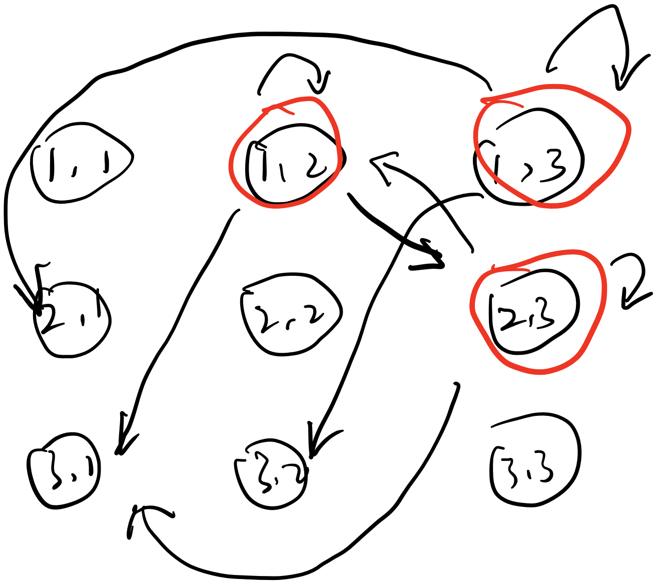
1 1

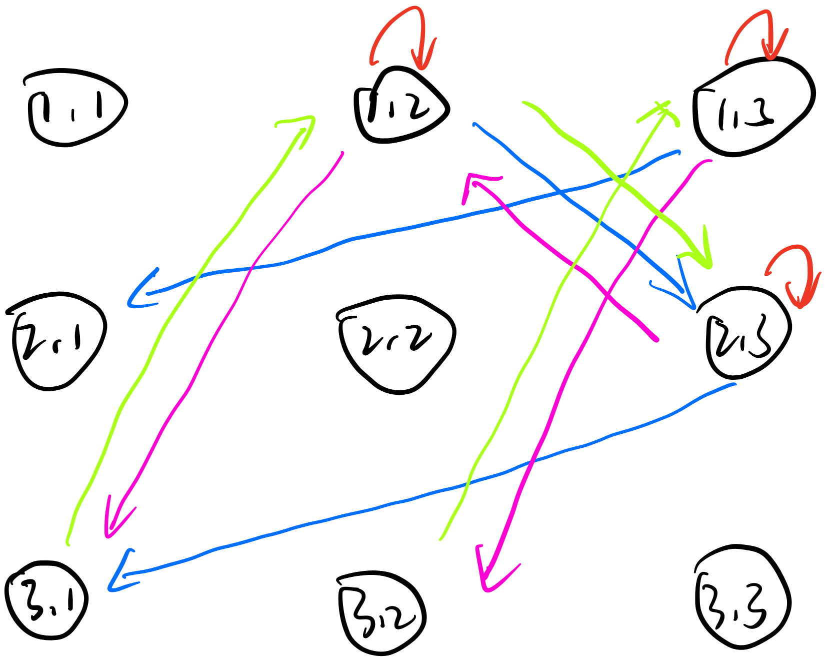
2 2

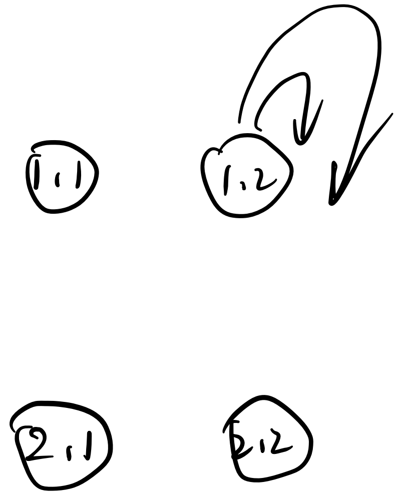


3 3

1 2 3
2 3 1
3 1 2

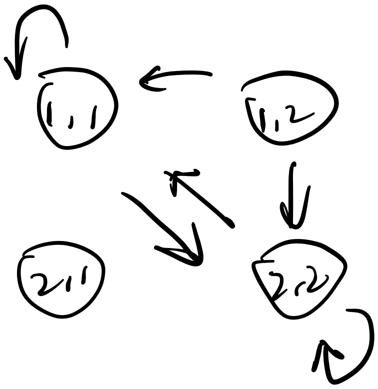






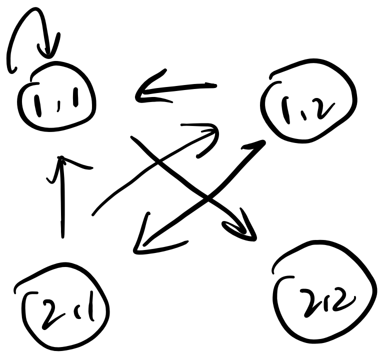
1 1

2 2



1 2

1 2



2	1
1	1

Sink should have k, k

another way to the sink

try

adjacency matrix 2D array

dictionary

sink within

strongly connected component

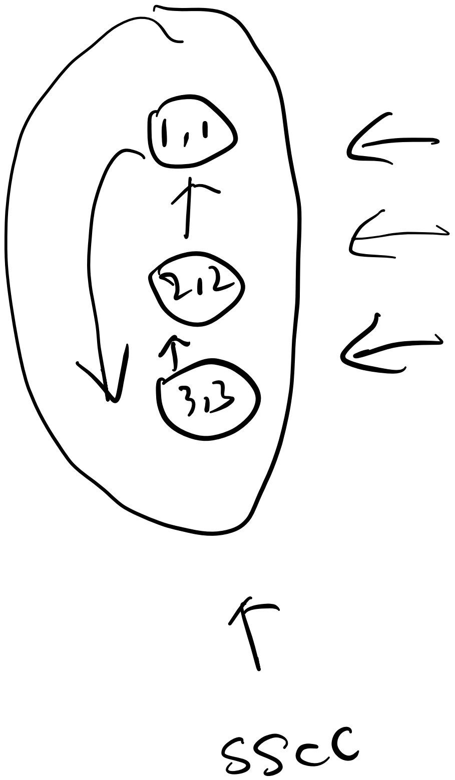
DFS ↗

SSCC

Run DFS on that sink in original graph

? sink must be only node in that SCC

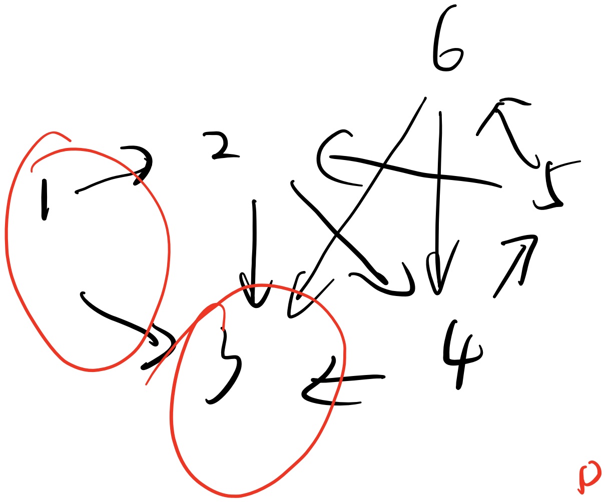
python array



running at every possible vertex

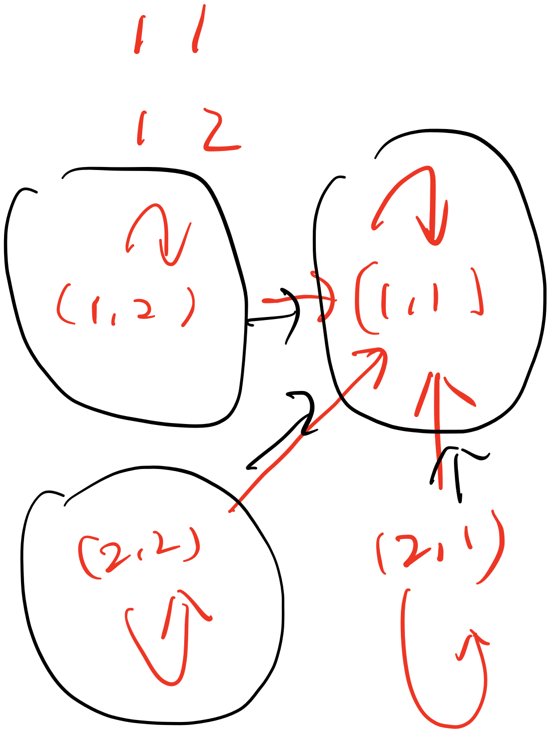
每个SCC中都有 1个(k,k)

那些点都跑到过



[[( ),( )],

DFSC 终点,



[( ) : [( ), ( )]]

1 1
1 2

i=1 j=0

(0,1) (0,0)

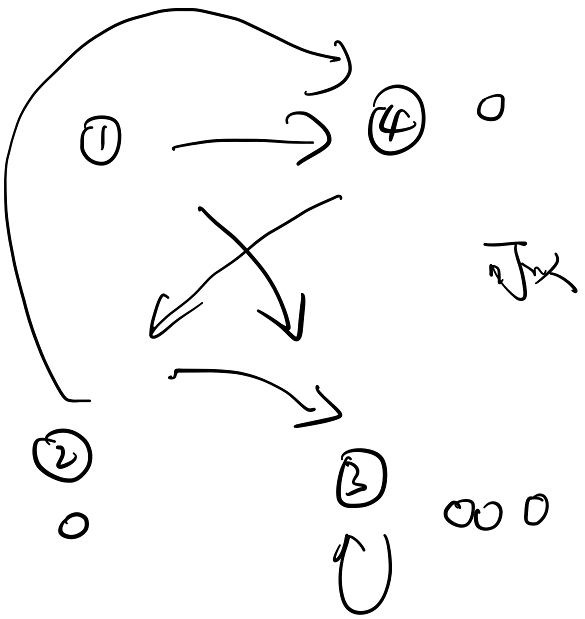
[0,0] [1,0]

(0,0) (0,1)

1

4 2

4	3
3	4
3	2
2	1



(1,1) (1,2) (1,3) (1,4)

(1,1) (1,2) (1,3) (1,4)

(2,1) (2,2) (2,3) (2,4)

(3,1) (3,2) (3,3) (3,4)

(4,1) (4,2) (4,3) (4,4)

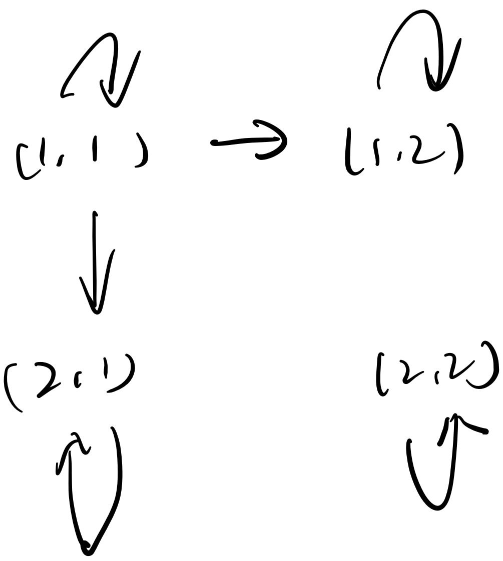
对所有(k,k)点跑DFS　跑一个点标记一个点

最后看是不是所有点都被标记了

$$\begin{matrix} 1 & 1 \\ 1 & 2 \end{matrix}$$

(1,1) → (1,1)

(1,2) → (1,1)　(1,2) → (1,2)



(j,i) → [j][k]

[i][k]

i →

j →

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)