Assignment 1

id: 519021910861

name: huidong xu

`date: 2021-10-12`

3-2

2. Give an algorithm to detect whether a given undirected graph contains a cycle. If the graph contains a cycle, then your algorithm should output one. (It should not output all cycles in the graph, just one of them.) The running time of your algorithm should be $O(m + n)$ for a graph with n nodes and m edges.

检测一个无向图是否有环，即检测这是否是一颗树，但由于增加了要求：如果有环就要输出其中任意一个环，所以我选用 DFS 算法，从任一节点开始，深度搜索它的相邻节点，并增加一个集合记录节点是否被遍历过。如果遍历某节点时发现该节点已经被遍历过一次，则回溯输出这条带环路径。由于该算法最多只会遍历 n 个节点和 m 条边，所有时间复杂度为 O(m + n)。具体算法伪代码如下：

```
set<node> isTraverse <- empty
size <- the number of nodes
_find(node root, vector<node>& v)
    /* 如果全部结点都已经被遍历过一次则返回 false */
    if isTraverse.size == size: return false
    /* 如果当前节点已经在遍历集合中则找到该环 */
    if root is in isTraverse:
        return true
    /* 将当前节点加入遍历集合中 */
    isTraverse.add(root)
    v.push_back(root)
    /* 如果当前节点的相邻节点还有没有被遍历过的 */
```

```
    while some node adjacent to root is not in isTraverse:
        node cur <- choose one of these nodes
        if (_find(cur, v) == true) return true
    /* 还原 vector 状态*/
    v.pop_back()

main:
    vector<root> v <- empty
    if find(root, v) == true: return v;
    else return false
```

3-4

4. Inspired by the example of that great Cornellian, Vladimir Nabokov, some of your friends have become amateur lepidopterists (they study butterflies). Often when they return from a trip with specimens of butterflies, it is very difficult for them to tell how many distinct species they've caught—thanks to the fact that many species look very similar to one another.

One day they return with n butterflies, and they believe that each belongs to one of two different species, which we'll call A and B for purposes of this discussion. They'd like to divide the n specimens into two groups—those that belong to A and those that belong to B—but it's very hard for them to directly label any one specimen. So they decide to adopt the following approach.

For each pair of specimens i and j, they study them carefully side by side. If they're confident enough in their judgment, then they label the pair (i, j) either "same" (meaning they believe them both to come from the same species) or "different" (meaning they believe them to come from different species). They also have the option of rendering no judgment on a given pair, in which case we'll call the pair *ambiguous*.

So now they have the collection of n specimens, as well as a collection of m judgments (either "same" or "different") for the pairs that were not declared to be ambiguous. They'd like to know if this data is consistent with the idea that each butterfly is from one of species A or B. So more concretely, we'll declare the m judgments to be *consistent* if it is possible to label each specimen either A or B in such a way that for each pair (i, j) labeled "same," it is the case that i and j have the same label; and for each pair (i, j) labeled "different," it is the case that i and j have different labels. They're in the middle of tediously working out whether their judgments are consistent, when one of them realizes that you probably have an algorithm that would answer this question right away.

Give an algorithm with running time $O(m + n)$ that determines whether the m judgments are consistent.

检测 m 条语句是否会造成冲突，由于语句会宣称两个节点是一致的或相异的，所以我们可以以此为依据构造图，“一致性”语句中两个节点即连通，“相异性”语句中某个节点放在另外一个图中，随后判断是否会有语句造成冲突。简单起见，我们可以用并查集来表示这两个子图，具体实践中可以直接更换成两个集合来表示两个子图中的结点。由于算法最多会遍历 n 个节点的 m 条边，所以算法复杂度为 O(m + n)。具体算法伪代码如下：

```
set<node> A <- empty, B <- empty
set<judgement> isAssure <- empty
size <- the number of judgements
while isAssure.size != size:
    choose one of judgement as the begining:
```

```
        node_a, node_b <- two different nodes of judgement_i
        /* 将这条语句的两个节点根据语义随机分配到 set 中 */
        if judgement_i is "same":
            A.add(node_a); B.add(node_b);
        else if judgement_i is "different":
            A.add(node_a); B.add(node_b);
    if isAssure.size == size: return true

    flag <- false /* 记录本次过程中是否有语句被确认 */
    do:
        judgement_i <- one of judgements not in isAssure
        node_a, node_b <- two different nodes of judgement_i
        if judgement_i is "same":
            /* 如果两个节点在相异的图中 */
            if node_a is in A && node_b is in B: return
false
            if node_a is in B && node_b is in A: return
false
            /* 集合具有唯一性，所以重复添加不影响结果 */
            if node_a is in A: A.add(node_b), flag = true
            else if node_a is in B: B.add(node_b), flag =
true
            else if node_b is in A: A.add(node_a), flag =
true
            else if node_b is in B: B.add(node_a), flag =
true
            /* 如果两个节点都还不在集合中，则等待 */
            else continue
        else if judgement_i is "different":
            /* 如果两个节点在相同的图中 */
            if node_a is in A && node_b is in A: return
false
            if node_a is in B && node_b is in B: return
false
            /* 将两个节点添加到不同的图中 */
            if node_a is in A: B.add(node_b), flag = true
            else if node_a is in B: A.add(node_b), flag =
true
```

```
            else if node_b is in A: B.add(node_a), flag =
true
            else if node_b is in B: A.add(node_a), flag =
true
            /* 如果两个节点都还不在集合中，则等待 */
            else continue
        while (flag)

return true
```

3-7

7. Some friends of yours work on wireless networks, and they're currently studying the properties of a network of n mobile devices. As the devices move around (actually, as their human owners move around), they define a graph at any point in time as follows: there is a node representing each of the n devices, and there is an edge between device i and device j if the physical locations of i and j are no more than 500 meters apart. (If so, we say that i and j are "in range" of each other.)

They'd like it to be the case that the network of devices is connected at all times, and so they've constrained the motion of the devices to satisfy the following property: at all times, each device i is within 500 meters of at least $n/2$ of the other devices. (We'll assume n is an even number.) What they'd like to know is: Does this property by itself guarantee that the network will remain connected?

Here's a concrete way to formulate the question as a claim about graphs.

Claim: Let G be a graph on n nodes, where n is an even number. If every node of G has degree at least $n/2$, then G is connected.

Decide whether you think the claim is true or false, and give a proof of either the claim or its negation.

正确。反证法。假设图中有 n 个节点，n 大于等于 2 且是偶数。不妨取任一节点记为 node_a，则该节点将与图中另外 n / 2 个节点连通，即由 1 + n / 2 个节点是连通的。假设存在某个节点不与上述 1 + n / 2 个节点连通，根据题设：每个节点都需要有 n / 2 个连通节点，又剩余的节点数量是$n-(1+n/2)-1=n/2-2<n/2$，与题设矛盾。故假设不成立。结论是正确的。

date: 2021-10-15

4-1

1. Decide whether you think the following statement is true or false. If it is true, give a short explanation. If it is false, give a counterexample.

Let G be an arbitrary connected, undirected graph with a distinct cost $c(e)$ on every edge e. Suppose e^* is the cheapest edge in G; that is, $c(e^*) < c(e)$ for every edge $e \neq e^*$. Then there is a minimum spanning tree T of G that contains the edge e^*.

真。交换论证。假设不存在包含边 e* 的 G 的最小生成树，则对于 G 的任意一棵最小生成树，边 e* 的两个顶点 u 和 v 将通过另外一条路径连接。我们将 e* 添加到这棵最小生成树中，则构成了一个包含有 u 和 v 的环，所以我们去除这个环上任意一条除 e* 的边，既不会改变这棵树的连通性也不会增加这棵树的开销，即这个含有 e* 的边的生成树也是一棵最小生成树，故结论得证。

4-8

8. Suppose you are given a connected graph G, with edge costs that are all distinct. Prove that G has a unique minimum spanning tree.

易证，图 G 中权值最小的边一定是最小生成树的边。（否则最小生成树加上权值最小的边后构成一个环，去掉环中任意一条非此边则形成了另一个权值更小的生成树）

设 T，T' 为 G 的俩个最小生成树，设 T 的边集 E(T) = {e1, e2, … , em}, T' 的边集 E(T') = {e'1, e'2, …, e'm}。设 ek 满足 ek != e'k 且 k 最小，由于所有边权值不同，不妨假设 weight(ek) < weight(e'k)，则将 ek 加入到 T'，T' 中构成环，易知环中不包含 e'1, e'2, …, e'k-1（否则在 T 中有包含 ek 的环），将环中任意非 ek 边删除后得到了权值更小的生成树，这与 T' 为最小生成树相矛盾，故 G 最小生成树唯一。

4-9

9. One of the basic motivations behind the Minimum Spanning Tree Problem is the goal of designing a spanning network for a set of nodes with minimum *total* cost. Here we explore another type of objective: designing a spanning network for which the *most expensive* edge is as cheap as possible.

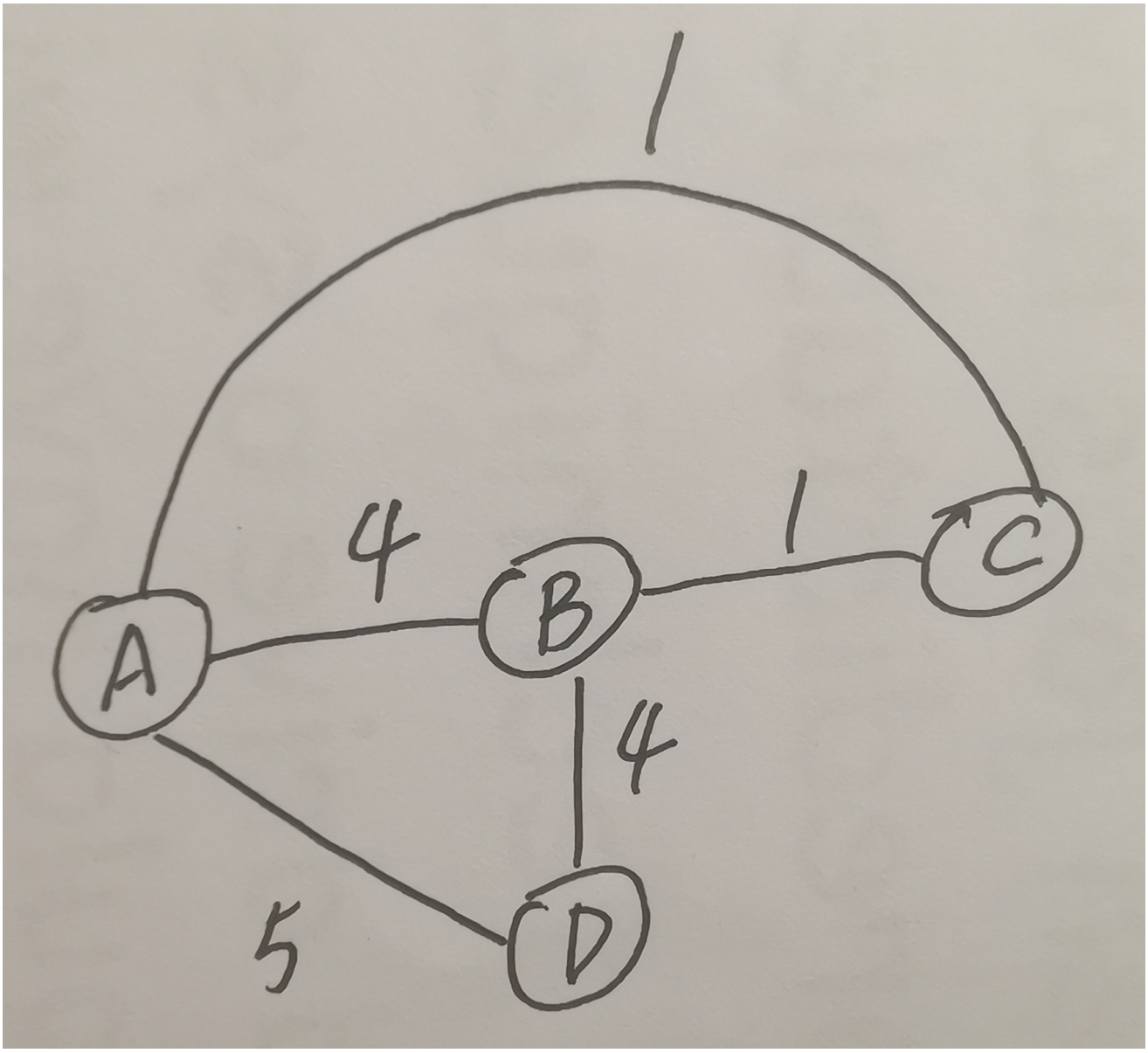
Specifically, let $G = (V, E)$ be a connected graph with n vertices, m edges, and positive edge costs that you may assume are all distinct. Let $T = (V, E')$ be a spanning tree of G; we define the *bottleneck edge* of T to be the edge of T with the greatest cost.

A spanning tree T of G is a *minimum-bottleneck spanning tree* if there is no spanning tree T' of G with a cheaper bottleneck edge.

(a) Is every minimum-bottleneck tree of G a minimum spanning tree of G? Prove or give a counterexample.

(b) Is every minimum spanning tree of G a minimum-bottleneck tree of G? Prove or give a counterexample.

两个结论均不成立。如下图：
最小瓶颈生成树边的权值分别是 4-4-1。
而最小生成树边的权值分别是 5-1-1。



4-29

29. Given a list of n natural numbers $d_1, d_2, \ldots, d_n$, show how to decide in polynomial time whether there exists an undirected graph $G = (V, E)$ whose node degrees are precisely the numbers $d_1, d_2, \ldots, d_n$. (That is, if $V = \{v_1, v_2, \ldots, v_n\}$, then the degree of v_i should be exactly d_i.) G should not contain multiple edges between the same pair of nodes, or "loop" edges with both endpoints equal to the same node.

我们可以结合贪心算法和深度优先算法来进行查询。首先将该列表从大到小进行排序，维护两个集合，集合 S 用于记录已经选择的点，集合 S' 用于记录不能被选择的点，初始化 S 和 S' 均为空。取出列表第一个值 n（即当前最大值），从图 G 中选择出每个度数大于等于该值的点集，并每次从中选出一个点进行计算。对选出的点的 m 条边，每次挑出 n 条边，将这些边所连接的点加入集合 S 中，并将其他边所连接的点加入集合 S'。之后每次搜索都从集合 S 中挑出度数满足条件的点进行操作，如

果在搜索过程中无法选出这样的点，则回溯。如果在搜索过程中整个列表被遍历完成，则存在这样的无向图，否则不存在。