# CS-600-A Homework 7

Zijing Huang 10414952

**R-13.7 Would you use the adjacency list structure or the adjacency matrix structure in each of the following cases? Justify your choice.**

1. **The graph has 10,000 vertices and 20,000 edges, and it is important to use as little space as possible.**

I would choose adjacency list structure in this case, because it saves space. If we use adjacent matrix, we will need a 10,000\*10,000 array to store only 20,000 edges.

1. **The graph has 10,000 vertices and 20,000,000 edges, and it is important to use as little space as possible.**

Either using adjacent list or matrix is fine in this case. Both of them matches the space requirement. But the adjacency matrix is good at judging whether two vertexes are adjacent and the adjacency list is good at inserting vertex and removing vertex.

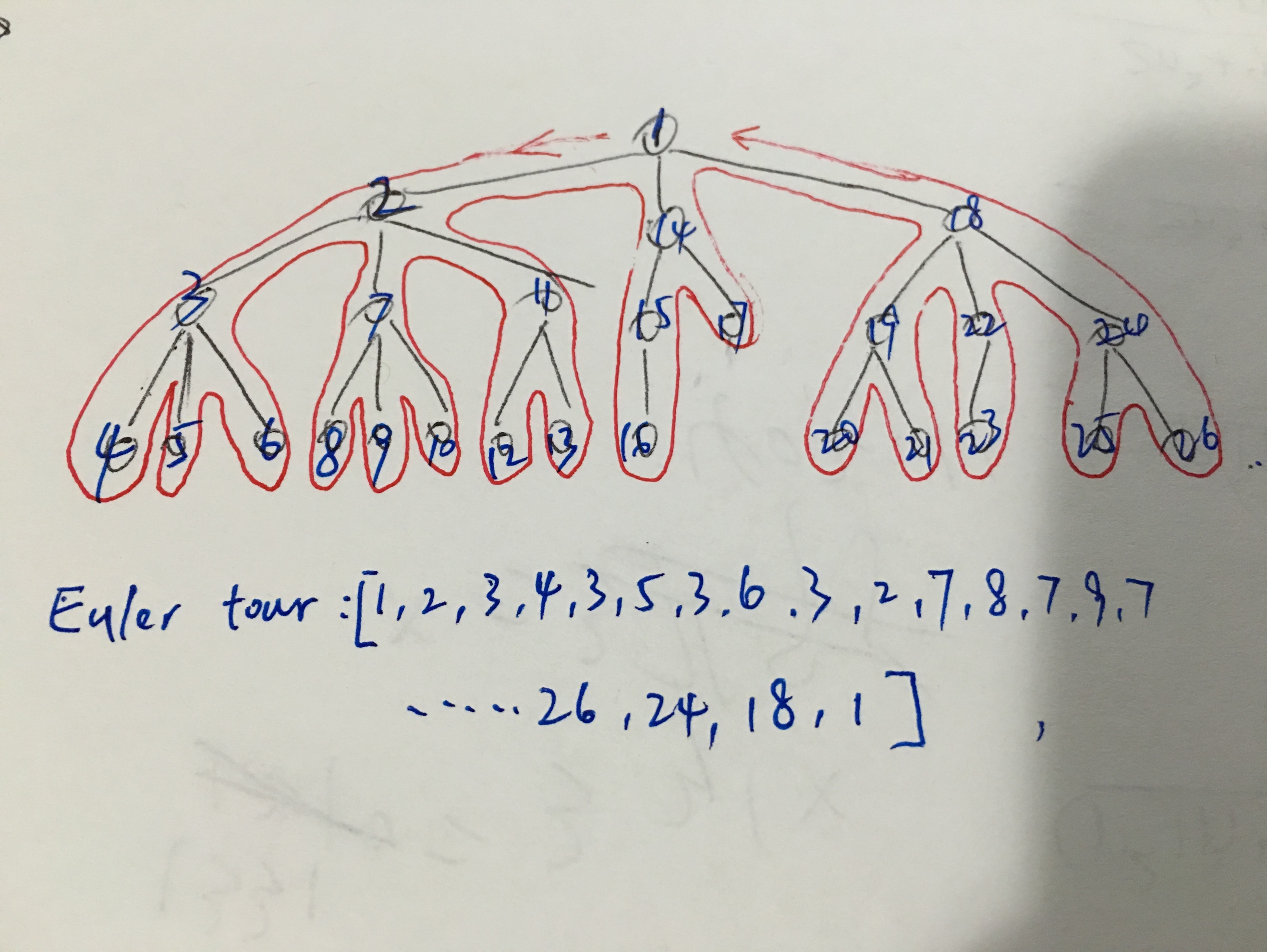
1. **You need to answer the query are Adjacent as fast as possible, no matter how much space you use.**

As the answer above, the adjacency matrix is good at judging whether two vertexes are adjacent. So we use adjacency matrix in this case to get a better running time, which is O(1) time.

**C-13.11 The directed version of the BFS algorithm classifies nontree edges as being either back edges or cross edges, but it does not distinguish between these two types. Given a BFS spanning tree, T, for a directed graph, Gì , and a set of nontree edges, E , describe an algorithm that can correctly label each edge in E as being either a back edge or cross edge. Your algorithm should run in O(n + m) time, where n is the number of vertices and m is the number of edges.**

***Hint:* Consider first constructing an Euler tour traversal of the tree T.**

As the hint say, we can build an Euler tour traversal of T first. We can differentiate back edges from cross edges by judging an edge(a,b) whether b is the ancestor of a. If yes, it is a back edge. We label each vertex Fv when first visit for v and Lv when last visit v. If the edge [Fw, Lw] is contained inside [Fv, Lv], we can say v is an ancestor of w, and this edge(w, v) is a back edge. For example, we have an edge(6,2) and we have a subsequence 6,3,2 in the Euler tour as follows. So the edge(6,2) is a back edge. Each differentiation cost O(1) time, and the total running time is O(n+m).



**A-13.6 A company named RT&T has a network of n stations connected by m high-speed communication links. Each customer’s phone is connected to one station in his or her area. The engineers of RT&T have developed a prototype video-phone system that allows two customers to see each other during a phone call. In order to have acceptable image quality, however, the number of links used to transmit video signals between the two parties cannot exceed 4. Suppose that RT&T’s network is represented by a graph. Design an efficient algorithm that computes, for each station, the set of stations it can reach using no more than 4 links.**

We can use a modified BFS traversal for each vertex and then we stop when the level of the computing of this vertex reaches 4.

**C-14.7 Suppose you are given a connected weighted undirected graph, G, with n vertices and m edges, such that the weight of each edge in G is an integer in the interval [1, c], for a fixed constant c > 0. Show how to solve the single-source shortest-paths problem, for any given vertex v, in G, in time O(n + m).**

***Hint:* Think about how to exploit the fact that the distance from v to any other vertex in G can be at most O(cn) = O(n).**

Let a priority queue, Q, contain all the vertices of G using the D labels as keys (D[u] is the distance from v to u in G). And let it to be a lookup table T, where T[i] means all the vertices that D[v] is equal to i. Since while we use Dijkstra’s algorithm, the distance of D will be monotonically increasing through each iteration. Using the amortization method, we can always keep track of T[i] in O(1) time while i is the smallest index. So we can use the profit to perform removeMin and update in O(1) time. Therefore, the total running time is O(n+m) in this case.

**A-14.2 Suppose that CONTROL, a secret U.S. government counterintelligence agency based in Washington, D.C., has build a communication network that links n stations spread across the world using m communication channels between pairs of stations. Suppose further that the evil spy agency, KAOS, is able to eavesdrop on some number, k, of these channels and that CONTROL knows the k channels that have been compromised. Now, CONTROL has a message, M, that it wants to send from its headquarters station, s, to one of its field stations, t. The problem is that the message is super secret and should traverse a path that minimizes the number of compromised edges that occur along this path. Explain how to model this problem as a shortest-path problem, and describe and analyze an efficient algorithm to solve it.**

We can simply assume that the weight of each uncompromised edge is 0 and the weight of the compromised edge is 1. And the goal is to find the minimizes number of compromised edges, so we can apply the Dijkstra’s algorithm to find the shortest path, which runs in O((n+m)\* logn) time.

**A-14.5 As your reward for saving the Kingdom of Bigfunnia from the evil monster “Exponential Asymptotic,” the king has given you the opportunity to earn a big reward. Behind the castle there is a maze, and along each corridor of the maze there is a bag of gold coins. The amount of gold in each bag varies. You will be given the opportunity to walk through the maze, picking up bags of gold. You may enter only through the door marked “ENTER” and exit through the door marked “EXIT.” (These are distinct doors.) While in the maze you may not retrace your steps. Each corridor of the maze has an arrow painted on the wall. You may only go down the corridor in the direction of the arrow. There is no way to traverse a “loop” in the maze. You will receive a map of the maze, including the amount of gold in and the direction of each corridor. Describe and analyze an efficient algorithm to help you pick up the most gold in this maze while traversing a path from the start to the finish.**

Make a directed acyclic graph base on the map. Add the weight of each edge base on the amount of gold. And then we can apply Dijkstra’s shortest-path algorithm for this problem. But instead finding the shortest path, we find the most weighted path in this case, which is the one with the most gold.