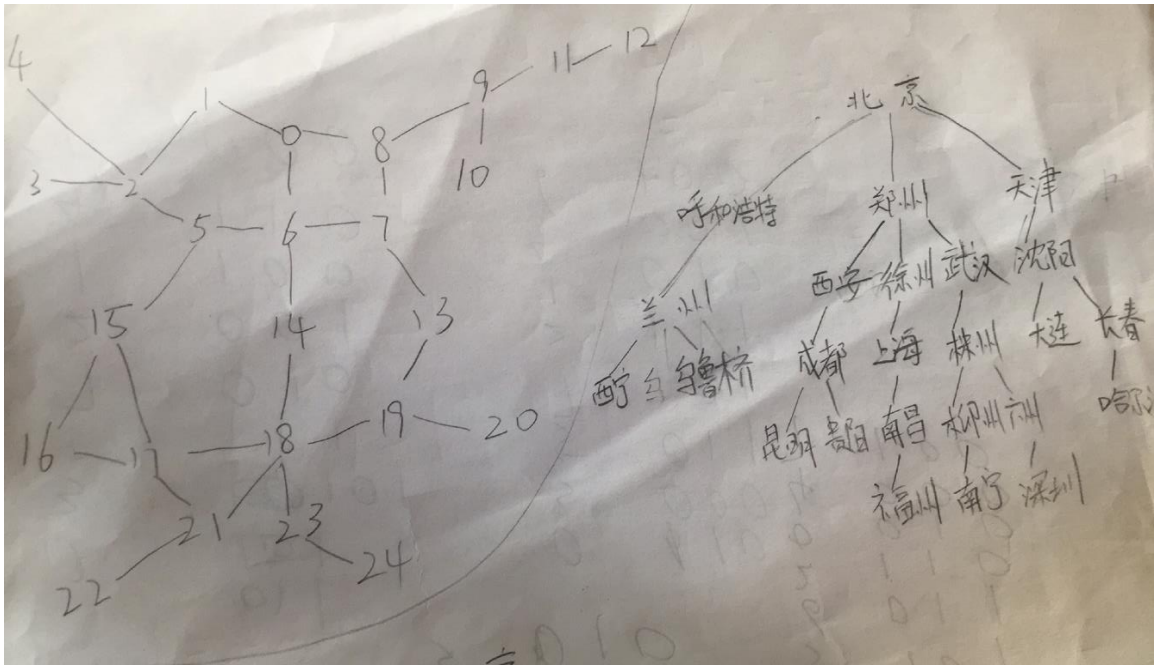
PROJECT 4

本人的代码直接将样例在内部函数输入，起点为北京，所以没有输入。按照下图编号：



函数说明

void path(int vex1 , int vex2 , graph* g, int weight)

在 vex1 和 vex2 之间建立边；
采用尾插法，每次找一个点已经连上的最后一条边，再将新建的边插入最后；

graph* creategraph()

图的创建函数，调用 path 函数默认创建的图为上图所示；

void BFS(graph *g , int v , string *place)

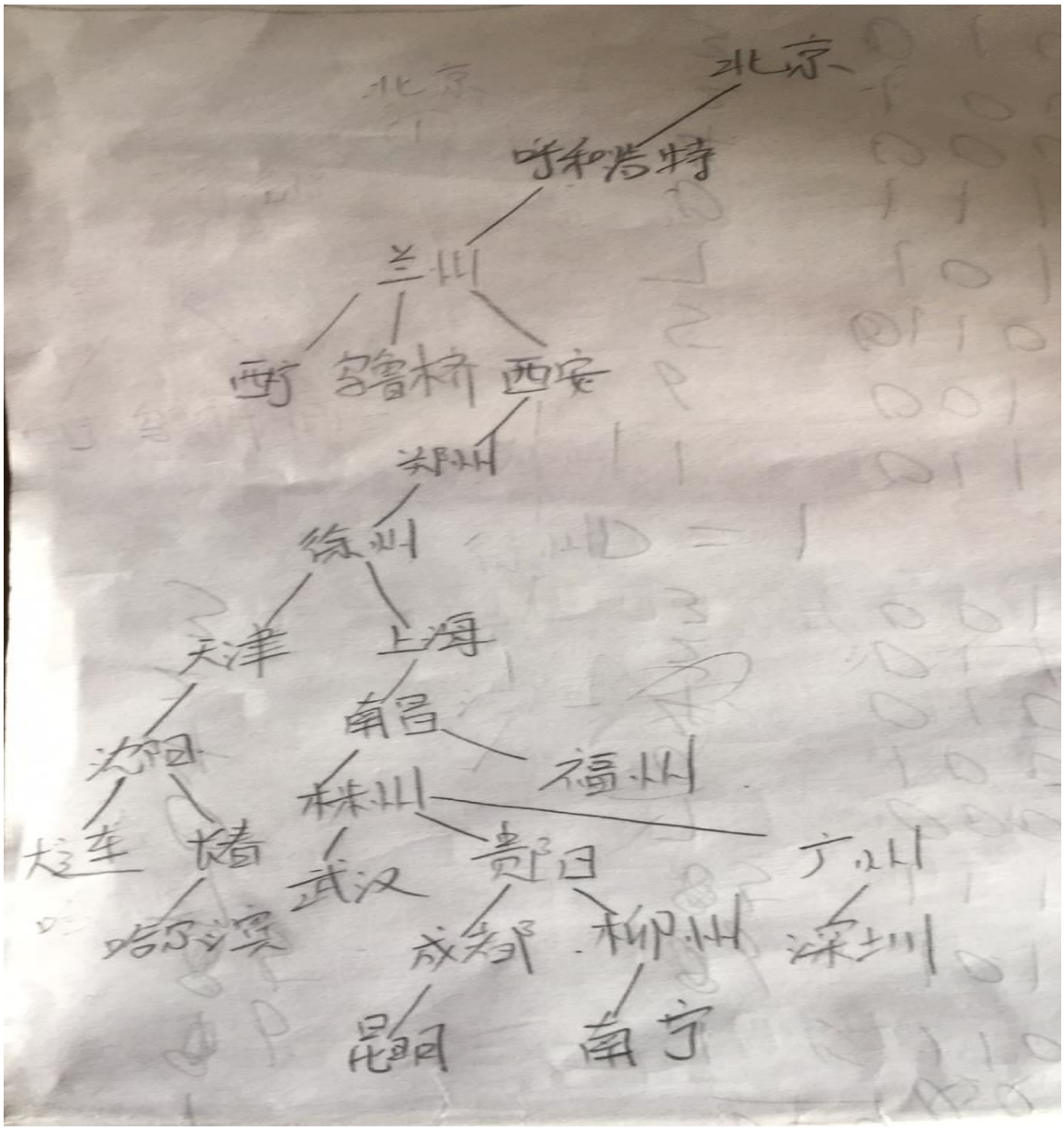
广度优先遍历函数，用队列模拟，每得到队列的最开头结点，则遍历该节点周围的节点并且将该节点压入队列中，直至队列为空；

void DFS(graph *g , int v , string *place)

深度优先搜索函数，采用栈模拟，没得到栈的顶头节点，则遍历该节点周围第一个还没被遍历的节点，并且压入栈中；如果该节点周围节点全部被遍历完，则将该节点退出栈。直至栈为空；

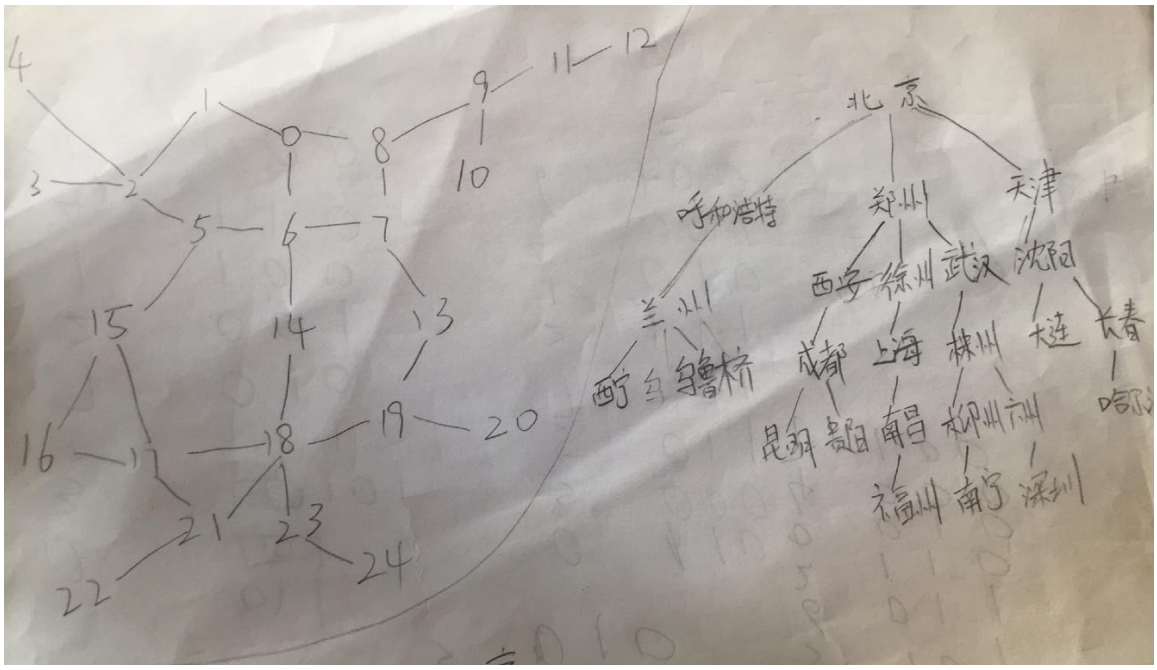
Vexnode* DSPT(graph* g , int v , string* place)

由于本人对题目的理解为用邻接表存储生成树，所以采用邻接表为树的存储结构，生成树的过程与深度优先搜索过程类似，每遍历一个点则与上一个点建立父亲孩子关系；生成树的结构为：



Vexnode* BSPT(graph* g , int v , string* place)

由于本人对题目的理解为用邻接表存储生成树，所以采用邻接表为树的存储结构，生成树的过程与广度优先搜索过程类似，每遍历一个点则与上一个点建立父亲孩子关系；生成树的结构为：



void Preorder(Vexnode *st , int v , int *visit,string * place)

对树的前序遍历，输出节点顺序；采用递归方法，每次访问一个点，输出该节点并且继续访问该节点的第一棵没访问的子树；

void Preorder_edge(Vexnode *st , int v , int *visit,string * place)

对树的前序遍历，输出访问的边的顺序；

void Inorder(Vexnode *st , int v , int *visit , string *place)

对生成树的中序遍历，输出节点顺序，采用递归方法，每次访问一个节点，若他的第一个孩子的子树已经被访问过，或者没有孩子，则输出该节点，如果有第二个孩子，则访问第二个孩子。

void Inorder_edge(Vexnode *st , int v , int *visit , string *place)

对生成树的中序遍历，输出边的访问顺序；

void Postorder(Vexnode *st , int v , int *visit , string *place)

对生成树的后序遍历，输出点的访问顺序；采用递归方法，假如一个节点没有孩子或者他的孩子都被访问完了，则输出该节点；

void Postorder_edge(Vexnode *st , int v , int *visit , string *place)

对生成树的后序遍历 ， 边的输出 ， 没有访问的边不输出

void buildtree(Vexnode*st , int v , tnode* root)

因为打印树采用凹入法，为了方便打印，此函数将用邻接表存储的树转化为用兄弟孩子树存储；

void printtree(tnode* root , string* place)

采用递归方法，用凹入法打印树；