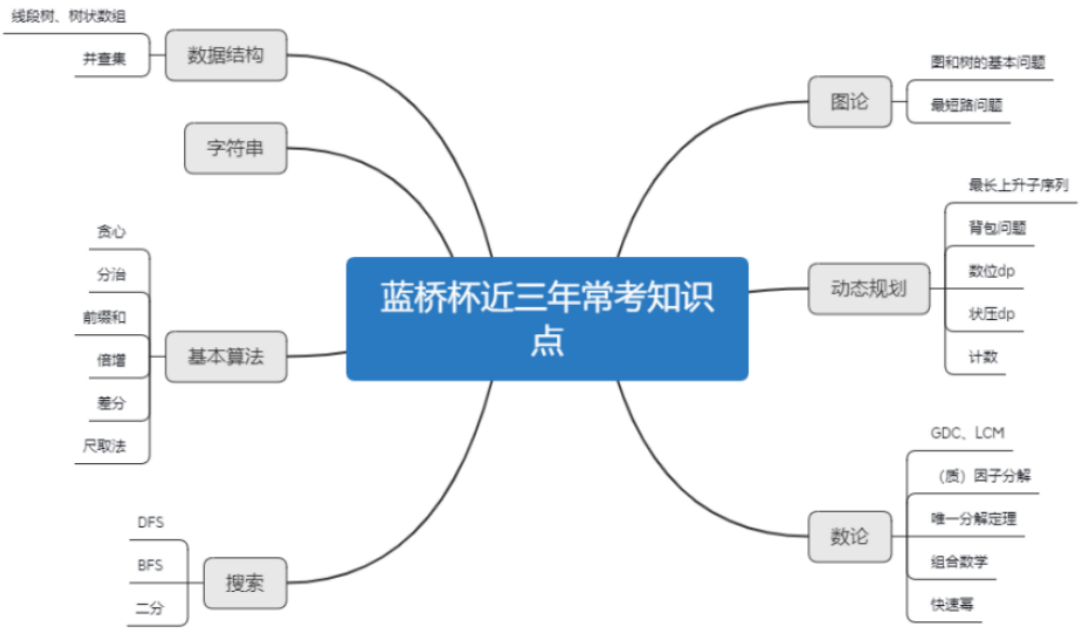
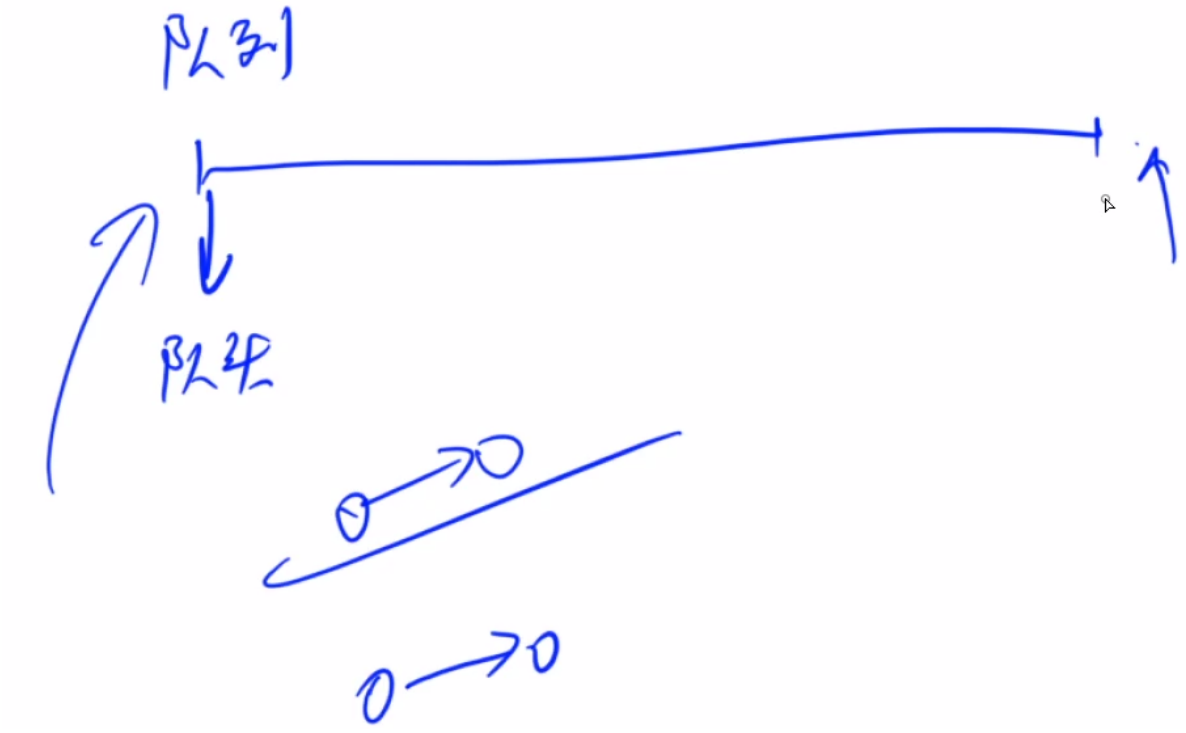


1、BFS

BFS中的多源BFS 求当前点到最近的目标点的距离

> 初始化的时候先把所有的目标源点放入队列，算出离他最近的起点的距离

双端队列



> 插入的时候，边权为正的点放在队尾 边权为0的点放到队头，任何可以转换为dijkstra算法的算法都是正确的，这就是一个特殊的dijkstra算法

双向BFS

> 每次扩展的时候，选择队列A或者队列B 但是每次要把整层的点都扩展，不能只扩展一个点

2. dfs

深搜什么时候需要恢复现场

恢复现场就是把操作逆一下

1.棋盘内部搜索

2.整个棋盘当作一种状态进行搜索

dfs连通性模型

> 内部搜索 不需要恢复现场
>
> 外部搜索 需要恢复现场
>
> 如果是从身体内部搜到另外一个地方是不需要恢复现场的，每个点只会被遍历一次，如果是把人当卓一个整体，变成另一个人，则需要恢复现场
>
> 整体的话每次搜索要保证开始状态是一样的

- dfs搜索顺序

dfs剪枝与优化

> 优化搜索顺序
>
> 尽量搜索分支少的节点
>
> 排除等效冗余　选组合不选排列
>
> 可行性剪枝
>
> 最优性剪枝
>
> 记忆化搜索（DP）

搜索顺序 -> 优化和剪枝

- 迭代加深
- 双向dfs

3.targan算法-lca

求最近公共祖先

> 算法步骤
>
> 1.读入所有操作 存下每个点vector 的另一个点以及询问编号
>
> 2.初始化并查集
>
> 3.targan(root)
>
> a.设置当前点状态st为1
>
> b.如果和当前点u相邻的点没有递归回溯 则递归操作 操作完成记得将点j合并到u

> c.处理和u相邻点的操作，如果另一个点y已经是递归回溯的点了 则可以找到ans[i] 注意是ans[i]不是ans[u]
>
> d.当前点状态设置为2

```
void tarjan(int u)
{
    st[u] = 1;
    for(int i = h[u];i!=-1;i=ne[i])
    {
        int j = e[i];
        if(!st[j])
        {
            tarjan(j);
            f[j] = u;
        }
    }

    for(int i = 0;i<query[u].size();i++)
    {
        int y0 = query[u][i].first,node = query[u][i].second;
        if(st[y0] == 2)
        {
            ans[node] = find(y0);
        }
    }

    st[u] = 2;

}
int main()
{
    cin>>n;
    memset(h,-1,sizeof h);
    int a,b;
    int root;
    for(int i = 1;i<=n;i++)
    {
        cin>>a>>b;
        if(b == -1)
        {
            root = a;
        }else{
            add(a,b);
            add(b,a);
        }
    }
    for(int i = 1;i<=N;i++)
    {
        f[i] = i;
    }
    cin>>m;
    //int a,b;
    for(int i = 1;i<=m;i++)
    {
        cin>>a>>b;
        query[a].push_back({b,i});
```

```
        query[b].push_back({a,i});
        //残缺部分
        request[i] = {a,b};
    }
    tarjan(root);
    for(int i = 1;i<=m;i++)
       cout<<ans[i]<<endl;
    return 0;
}
```

快速乘

```
LL quick_mul(LL a,LL b,LL mod)//快速乘
{
    LL ans = 0;
    while(b)
    {
        if(b&1) ans = (ans+a)%mod;
        a = (a+a)%mod;
        b>>=1;
    }
    return ans;
}
```