49讲搜索：如何用A*搜索算法实现游戏中的寻路功能

魔兽世界、仙剑奇侠传这类MMRPG游戏，不知道你有没有玩过？在这些游戏中，有一个非常重要的功能，那就是人物角色自动寻路。**当人物处于游戏地图中的某个位置的时候，我们用鼠标点击另外一个相对较远的位置，人物就会自动地绕过障碍物走过去。玩过这么多游戏，不知你是否思考过，这个功能是怎么实现的呢？**

算法解析

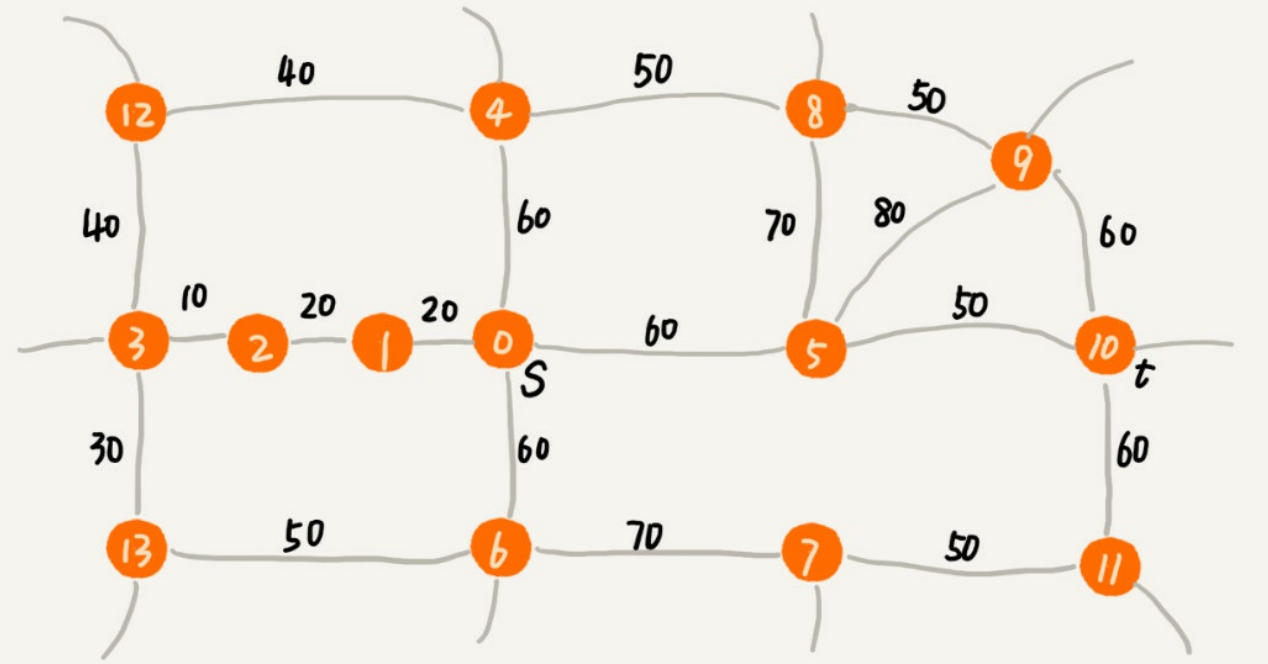
实际上，这是一个非常典型的搜索问题。人物的起点就是他当下所在的位置，终点就是鼠标点击的位置。我们需要在地图中，找一条从起点到终点的路径。这条路径要绕过地图中所有障碍物，并且看起来要是一种非常聪明的走法。所谓"聪明"，笼统地解释就是，走的路不能太绕。理论上讲，最短路径显然是最聪明的走法，是这个问题的最优解。

不过，在第44节最优出行路线规划问题中，我们也讲过，如果图非常大，那Dijkstra最短路径算法的执行耗时会很多。在真实的软件开发中，我们面对的是超级大的地图和海量的寻路请求，算法的执行效率太低，这显然是无法接受的。

实际上，像出行路线规划、游戏寻路，这些真实软件开发中的问题，一般情况下，我们都不需要非得求最优解（也就是最短路径）。在权衡路线规划质量和执行效率的情况下，我们只需要寻求一个次优解就足够了。那**如何快速找出一条接近于最短路线的次优路线呢？**

这个快速的路径规划算法，就是我们今天要学习的**A*算法**。实际上，A*算法是对Dijkstra算法的优化和改造。如何将Dijkstra算法改造成A*算法呢？为了更好地理解接下来要讲的内容，我建议你先温习下第44节中的Dijkstra算法的实现原理。

Dijkstra算法有点儿类似BFS算法，它每次找到跟起点最近的顶点，往外扩展。这种往外扩展的思路，其实有些盲目。为什么这么说呢？我举一个例子来给你解释一下。下面这个图对应一个真实的地图，每个顶点在地图中的位置，我们用一个二维坐标（x，y）来表示，其中，x表示横坐标，y表示纵坐标。



ID	0	1	2	3	4	5	6	7	8	9	10
x	320	300	280	270	320	360	320	370	350	390	400
y	630	630	625	630	700	620	590	580	730	690	620

在Dijkstra算法的实现思路中，我们用一个优先级队列，来记录已经遍历到的顶点以及这个顶点与起点的路径长度。顶点与起点路径长度越小，就越先被从优先级队列中取出来扩展，从图中举的例子可以看出，尽管我们找的是从s到t的路线，但是最先被搜索到的顶点依次是1，2，3。通过肉眼来观察，这个搜索方向跟我们期望的路线方向（s到t是从西向东）是反着的，路线搜索的方向明显“跑偏”了。

之所以会“跑偏”，那是因为我们是按照顶点与起点的路径长度的大小，来安排出队列顺序的。与起点越近的顶点，就会越早出队列。我们并没有考虑到这个顶点到终点的距离，所以，在地图中，尽管1，2，3三个顶点离起始顶点最近，但离终点却越来越远。

如果我们综合更多的因素，把这个顶点到终点可能还要走多远，也考虑进去，综合来判断哪个顶点该先出队列，那是不是就可以避免“跑偏”呢?

当我们遍历到某个顶点的时候，从起点走到这个顶点的路径长度是确定的，我们记作g(i)（i表示顶点编号）。但是，从这个顶点到终点的路径长度，我们是未知的。虽然确切的值无法提前知道，但是我们可以用其他估计值来代替。

这里我们可以通过这个顶点跟终点之间的直线距离，也就是欧几里得距离，来近似地估计这个顶点跟终点的路径长度（注意：路径长度跟直线距离是两个概念）。我们把这个距离记作h(i)（i表示这个顶点的编号），专业的叫法是**启发函数**（heuristic function）。因为欧几里得距离的计算公式，会涉及比较耗时的开根号计算，所以，我们一般通过另外一个更加简单的距离计算公式，那就是**曼哈顿距离**（Manhattan distance）。曼哈顿距离是两点之间横纵坐标的距离之和。计算的过程只涉及加减法、符号位反转，所以比欧几里得距离更加高效。

```
int hManhattan(Vertex v1, Vertex v2) { // Vertex表示顶点，后面有定义
  return Math.abs(v1.x - v2.x) + Math.abs(v1.y - v2.y);
}
```

原来只是单纯地通过顶点与起点之间的路径长度g(i)，来判断谁先出队列，现在有了顶点到终点的路径长度估计值，我们通过两者之和f(i)=g(i)+h(i)，来判断哪个顶点该最先出队列。综合两部分，我们就能有效避免刚刚讲的“跑偏”。这里f(i)的专业叫法是**估价函数**（evaluation function）。

从刚刚的描述，我们可以发现，A*算法就是对Dijkstra算法的简单改造。实际上，代码实现方面，我们也只需要稍微改动几行代码，就能把Dijkstra算法的代码实现，改成A*算法的代码实现。

在A*算法的代码实现中，顶点Vertex类的定义，跟Dijkstra算法中的定义，稍微有点儿区别，多了x，y坐标，以及刚刚提到的f(i)值。图Graph类的定义跟Dijkstra算法中的定义一样。为了避免重复，我这里就没有再贴出来了。

```
private class Vertex {
  public int id; // 顶点编号ID
  public int dist; // 从起始顶点，到这个顶点的距离，也就是g(i)
  public int f; // 新增：f(i)=g(i)+h(i)
  public int x, y; // 新增：顶点在地图中的坐标（x, y）
  public Vertex(int id, int x, int y) {
    this.id = id;
    this.x = x;
    this.y = y;
    this.f = Integer.MAX_VALUE;
    this.dist = Integer.MAX_VALUE;
  }
}
// Graph类的成员变量，在构造函数中初始化
Vertex[] vertexes = new Vertex[this.v];
// 新增一个方法，添加顶点的坐标
public void addVetex(int id, int x, int y) {
  vertexes[id] = new Vertex(id, x, y)
}
```

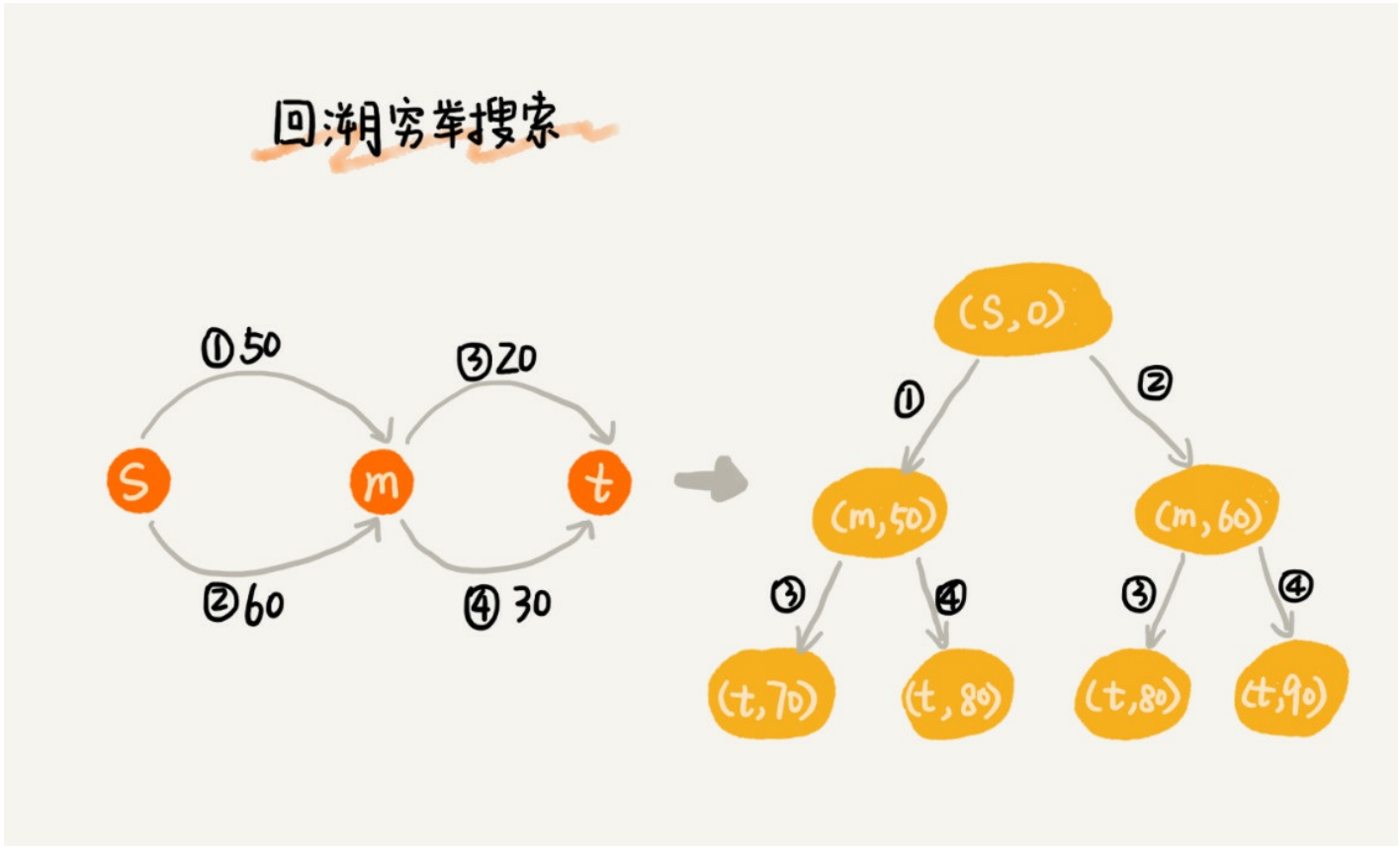
A*算法的代码实现的主要逻辑是下面这段代码。它跟Dijkstra算法的代码实现，主要有3点区别：

- 优先级队列构建的方式不同。A*算法是根据f值（也就是刚刚讲到的f(i)=g(i)+h(i)）来构建优先级队列，而Dijkstra算法是根据dist值（也就是刚刚讲到的g(i)）来构建优先级队列；
- A*算法在更新顶点dist值的时候，会同步更新f值；
- 循环结束的条件也不一样。Dijkstra算法是在终点出队列的时候才结束，A*算法是一旦遍历到终点就结束。

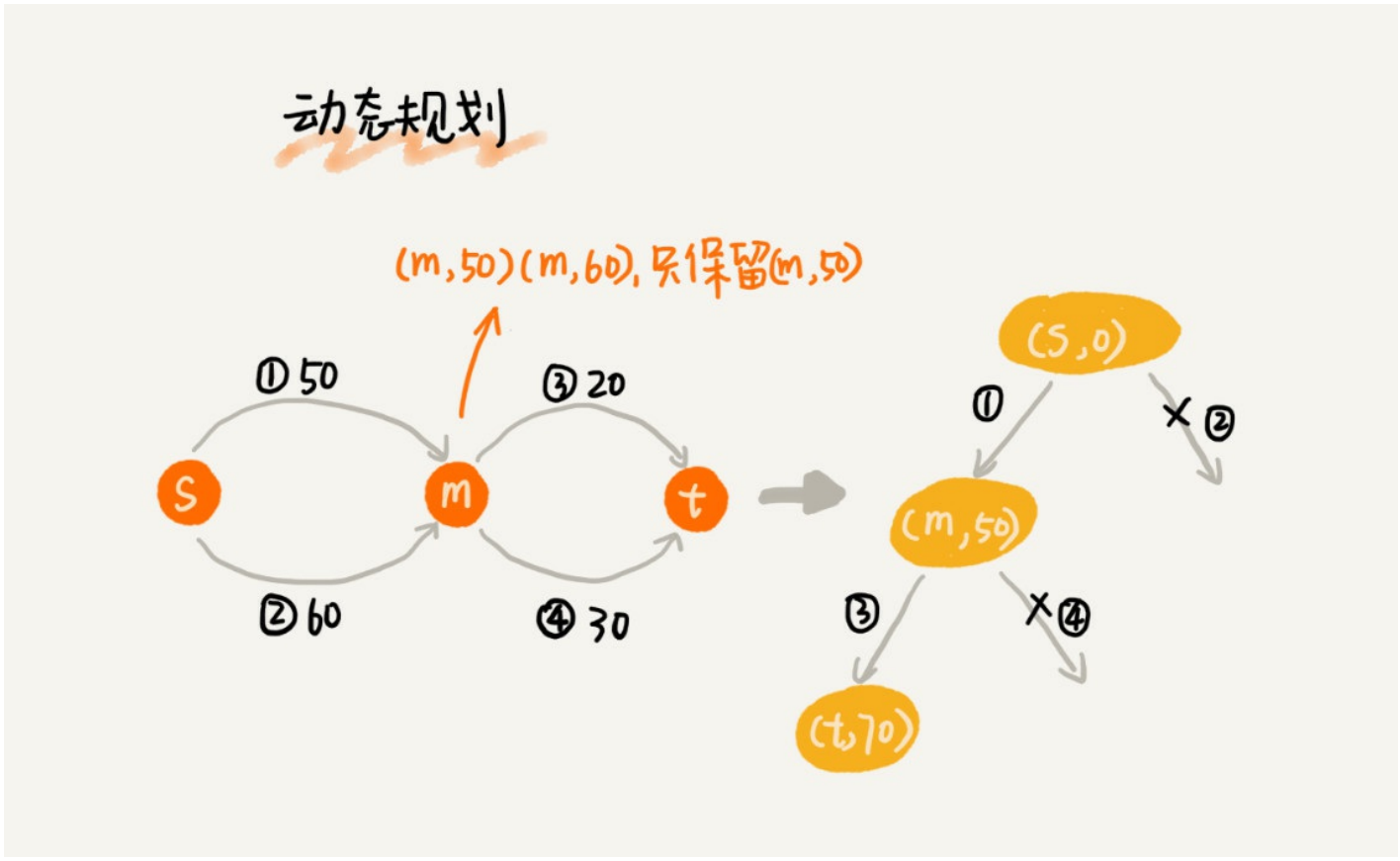
```
public void astar(int s, int t) { // 从顶点s到顶点t的路径
  int[] predecessor = new int[this.v]; // 用来还原路径
  // 按照vertex的f值构建的小顶堆，而不是按照dist
  PriorityQueue queue = new PriorityQueue(this.v);
  boolean[] inqueue = new boolean[this.v]; // 标记是否进入过队列
  vertexes[s].dist = 0;
  vertexes[s].f = 0;
  queue.add(vertexes[s]);
  inqueue[s] = true;
  while (!queue.isEmpty()) {
    Vertex minVertex = queue.poll(); // 取堆顶元素并删除
    for (int i = 0; i < adj[minVertex.id].size(); ++i) {
      Edge e = adj[minVertex.id].get(i); // 取出一条minVetex相连的边
      Vertex nextVertex = vertexes[e.tid]; // minVertex-->nextVertex
      if (minVertex.dist + e.w < nextVertex.dist) { // 更新next的dist,f
        nextVertex.dist = minVertex.dist + e.w;
        nextVertex.f
           = nextVertex.dist+hManhattan(nextVertex, vertexes[t]);
        predecessor[nextVertex.id] = minVertex.id;
        if (inqueue[nextVertex.id] == true) {
          queue.update(nextVertex);
        } else {
          queue.add(nextVertex);
          inqueue[nextVertex.id] = true;
        }
      }
      if (nextVertex.id == t) break; // 只要到达t就可以结束while了
    }
  }
  // 输出路径
  System.out.print(s);
  print(s, t, predecessor); // print函数请参看Dijkstra算法的实现
}
```

尽管A*算法可以更加快速的找到从起点到终点的路线，但是它并不能像Dijkstra算法那样，找到最短路线。这是为什么呢?

要找出起点s到终点t的最短路径，最简单的方法是，通过回溯穷举所有从s到达t的不同路径，然后对比找出最短的那个。不过很显然，回溯算法的执行效率非常低，是指数级的。



Dijkstra算法在此基础之上，利用动态规划的思想，对回溯搜索进行了剪枝，只保留起点到某个顶点的最短路径，继续往外扩展搜索。动态规划相较于回溯搜索，只是换了一个实现思路，但它实际上也考察到了所有从起点到终点的路线，所以才能得到最优解。



A*算法之所以不能像Dijkstra算法那样，找到最短路径，主要原因是两者的while循环结束条件不一样。刚刚我们讲过，Dijkstra算法是在终点出队列的时候才结束，A*算法是一旦遍历到终点就结束。对于Dijkstra算法来说，当终点出队列的时候，终点的dist值是优先级队列中所有顶点的最小值，即便再运行下去，终点的dist值也不会再被更新了。对于A*算法来说，一旦遍历到终点，我们就结束while循环，这个时候，终点的dist值未必是最小值。

A*算法利用贪心算法的思路，每次都找f值最小的顶点出队列，一旦搜索到终点就不在继续考察其他顶点和路线了。所以，它并没有考察所有的路线，也就不可能找出最短路径了。

搞懂了A*算法，我们再来看下，**如何借助A*算法解决今天的游戏寻路问题?**

要利用A*算法解决这个问题，我们只需要把地图，抽象成图就可以了。不过，游戏中的地图跟第44节中讲的我们平常用的地图是不一样的。因为游戏中的地图并不像我们现实生活中那样，存在规划非常清晰的道路，更多的是宽阔的荒野、草坪等。所以，我们没法利用44节中讲到的抽象方法，把岔路口抽象成顶点，把道路抽象成边。

实际上，我们可以换一种抽象的思路，把整个地图分割成一个一个的小方块。在某一个方块上的人物，只能往上下左右四个方向的方块上移动。我们可以把每个方块看作一个顶点。两个方块相邻，我们就在它们之间，连两条有向边，并且边的权值都是1。所以，这个问题就转化成了，在一个有向有权图中，找某个顶点到另一个顶点的路径问题。将地图抽象成边权值为1的有向图之后，我们就可以套用A*算法，来实现游戏中人物的自动寻路功能了。

总结引申

我们今天讲的A*算法属于一种**启发式搜索算法**（Heuristically Search Algorithm）。实际上，启发式搜索算法并不仅仅只有A*算法，还有很多其他算法，比如IDA*算法、蚁群算法、遗传算法、模拟退火算法等。如果感兴趣，你可以自行研究下。

启发式搜索算法利用估价函数，避免“跑偏”，贪心地朝着最有可能到达终点的方向前进。这种算法找出的路线，并不是最短路线。但是，实际的软件开发中的路线规划问题，我们往往并不需要非得找最短路线。所以，鉴于启发式搜索算法能很好地平衡路线质量和执行效率，它在实际的软件开发中的应用更加广泛。实际上，在第44节中，我们讲到的地图App中的出行路线规划问题，也可以利用启发式搜索算法来实现。

课后思考

我们之前讲的“迷宫问题”是否可以借助A*算法来更快速地找到一个走出去的路线呢？如果可以，请具体讲讲该怎么来做；如果不可以，请说说原因。

欢迎留言和我分享，也欢迎点击“请朋友读”，把今天的内容分享给你的好友，和他一起讨论、学习。

精选留言

传说中的成大大

今天看了A*算法 反而对dijkstra算法理解得更透彻了....

2019-01-18 11:12

yongxiang

王争老师，我把代码输入运行，并把过程打印出来，发现代码运行的过程跟您说的A*算法的三点区别中的第三点不一样，不会在遍历到目标顶点时退出while循环。您看是不是27行的break只是退出了for循环，无法退出while循环，是不是需要增加以下的修改：

```
if (nextVertex.id == t) {
queue.clear();
break;
}
```

2019-01-19 16:06

皇家救星

我记得以前看过的a*算法介绍还有close和open表，这里好像没提到?

2019-01-18 08:29

作者回复

那就是俩人造的概念 并没有太大意义。

2019-01-18 09:12

辰陌

请问一下老师，Astar算法，启发式距离的设置好像是有一定原则的，如果在满足一致性原则的基础之上，然后再抛除最后一步停止准则的影响的情况下，应该是可以找到最优解的吧?

2019-01-20 15:59

hua168

我之前是打算生管理，去个小公司，发现也要会开发，去年就毅然去学java，维护懂java会有帮助，也可以搞下大数据……再学一门本职运维开发需要python……

我就是这样打算的…

同学说我们学历低只要大专，问我要大家考研究生不？我感觉我不去大公司的话没什么用吧？但一想很多要求本科，自考研究

生不知道承认不？尤其公司，再说就算看完都老了吧……意义有多大？

2019-01-19 20:38

作者回复

看得到@hua168同学对职业规划很迷茫。

我来逐一回答一下你的问题：

1. 自考学历对你来说没用。绝大部分卡学历的公司，只看第一学历；不卡学历的那部分公司，你自考本科也没必要。自考学历对一小部分人有用，具体哪部分人适合我就不展开讲了，总之不适合你。但是，你没有因为学历自卑，公司这么多，总有不卡学历的。我见过很多大专文凭，技术去贼拉子好的，照样去大公司。

2. 不管是大公司还是小公司，都会卡年龄。不过所谓的卡年龄并不是说年龄大了就没人要了。而是能力跟年龄不符，年龄一大把却跟人家工作两三年经验能力差不多，要钱还贼高，那估计确实没人要。

3. 不要再去学java了。如果你还想走技术路线，那就要专精尖，这个我前一条回复说过了。

4. 我还是说了，对于技术一般的人来说，如果要升管理岗，还是那句话“要有领导气质”，另外，你要包装一下简历，一些很小公司的领导是识别不出来的：）听起来是不入流的建议，但是，我确实是认真的。

5. 实际上，年龄大了，技术没有太大竞争力，去个安稳的公司很好，比如国企性质的一些互联网保险公司，具体你自己搜搜吧，我这里不方便说公司名字。

以上建议只针对你本人的情况，并且是我的个人建议。如有不投，你自己斟酌。

2019-01-24 09:23

hua168

像我年龄35岁，学历是大专，您觉得有必须自考研究生之类的么？升级一下学历？自考类不知道去大公司是都承认？
英语不太好，只能勉强看懂……

2019-01-19 20:18

hua168

非常感谢……

2019-01-19 20:09

yongxiang

王争老师，这里的每条边的权重 w 跟两个顶点之间的 x，y 有相关关系吗？还是说可以随意定义？

2019-01-19 16:10

ALAN

老师，我在网上看到说a star算法启发函数选的好的话，估计路径比实际路径短的话，是可以找到最短路径的。老师，你能举个例子说明找到的不是最短路径吗，在估计路径比实践路径短的情况下？

2019-01-19 10:39

hua168

大神，能问一个题外话吗，关于自己人生规划，水平和眼界所限，想不通，
都说大神级见识很广也多，能给我这个35岁只维护过四五十台linux服务器的运维指条路吗？
现在很迷茫和压力大~~
能力如下：
一.网络：CCNA水平，自过了CCNP忘记了，当过2年网管
二、维护过asp.net电商网站，3年，只有简单的，兼职网管
三、linux运维，只在一家电商做了3年多，会
1.web：nginx、tomcat配置（少用）+php:nignx的rewirte和反代
2.数据库：mysql、mongoDB、redis 配置及主从，不会mycat、Cetus之类
3.反代：会nginx、haproxy简单配置
4.存储：NFS、fastDFS、hadoop简单看了一下

5.版本控制：只会git及搭建gitlab+jenkins（简单的CI/CD）
6.监控：简单配置zabbix+shell脚本
7.虚拟化：kvm安装及配置、docker(k8s还没学)
8.云计算：openstack只会安装做过实验
9.测试：只会ab工具
10.日志：ELK安装配置，还没结合java（在学中）
11.大数据：没使用过（不会flume、storm、spark、flink、kafka）
12.脚本：主要是shell为主、会点python

四、编程能力：自学，没项目经验
1.前端：
1）HTML（HTML5不怎看）
2）css（laiui、学了一下vue）
3) js、jquery框架、ES6简单看了一下
2.PHP：语法简单的thinkphp5框架
3.java：考虑要维护java web在学
只看了java、jsp及servet、spring、springMVC、spring Boot（这个为主）
4.python：考虑运维用到
python：会简单的脚本
django：只会官网简单的

问题是：现在已35岁了，失业，怎办？年龄摆在那里，能力好像不强，学历大专。
能给个建议吗？非常感谢~~
2019-01-19 00:43

作者回复

我下面说的话，可能会伤害到你，不过，我是非常认真的。

从你对运维相关的技术点的描述上，可以看出，你应该没有在一个稍微大点的公司工作过吧，所以，很多技术都用的不够深，都只是略知一二，没有自己拿得出手的东西。

建议你去稍微大点公司锻炼一下技术，同时，也能给你的履历加分。

不过，以你的年龄和履历，去稍微大点的公司可能也不现实了，因为现在好点的公司都卡学历、背景，更别说技术了。

所以，我建议你找一个运维领域的风口技术去研究，比如你提到的k8s。这种技术才兴起，会的人不多，所以招聘公司都不会太卡学历、经历，只要会，是个人都要，可以借机去个大点的公司。这会是你的一个转折点。

而且。现在，经济下行，互联网行业都压缩招聘。你正好利用这1、2年，沉下心来，抓住一个技术方向，研究深、研究透。

还有一条路，那就是做管理岗位。这个要看你有没有领导气质了：）如果有领导范，年龄大，工作经历多，也可以忽悠到一些小公司的管理岗。实际上，对你来说，这条路也是不错的。

还有一条路，那就是靠去天使轮的创业公司逆袭。这条路有点赌博的意思。不过，如果公司搞大了，你也会青云直上，这辈子都不愁了：）
2019-01-19 11:15

fy
老师，刷leetcode刷的有点苦逼，刷不动，你的专栏就看到排序就中断了，但都记好了笔记。
2019-01-18 22:42


纯洁的憎恶

对于有大片无变化的地形环境，是否可以采用更大的方块表示，同时增加其与邻接顶点的权值，已表示距离更远。这样可以减少顶点数，简化图的复杂程度，提高执行效率。不过可能造成行走路线中折线过多，不够平滑。

2019-01-18 18:45

蓝心

21行代码：queue.update(nextVertex); 是不是没有这个API 。要删除在插入呢

2019-01-18 10:54

『LHCY』

真实游戏中也是用的小方块来做的吗？比如要往(1，1)方向走，先把模型角度调整，然后移动是一个个小方格走的，因为方格太小使肉眼分辨不出？

2019-01-18 09:47