学习记录

王远皓

Abstract

本周学习内容：熟悉 LaTeX 的使用，制作了可供往后学习记录的模板，能提高些后面学习记录的效率；学习并了解一些最短路径算法，包括深度优先搜索算法 ***DFS***、广度优先搜索算法 ***BFS***、贪心算法、*A** 算法; 最后是在课上学习了用于简化矩阵方程求解的 *LU* 拆解。

1. LaTeX 个人使用手册 (CVPR 格式部分解析)

1.1. main.tex 部分功能解析

- \documentclas[10pt,twocolumn,letterpaper]{article}
  documentclass 定义文档类型。参数 [10pt,twocolumn,letterpaper] 表示字体大小为 10pt，双栏排版，纸张大小为 letter（美国标准大小）。
- \usepackage 用来加载 cvpr 宏包。
  \usepackage[review]{cvpr} 表示将生成用于评审（review）的版本，会带有行号。
  \usepackage{cvpr} 是最终提交时的版本，不带行号。
  \usepackage[pagenumbers]{cvpr} 添加页码，一般用于预印版（如 arXiv）。
- \usepackagesetspace
  单倍行距
  \singlespacing
  1.5 倍行距
  \onehalfspacing
  双倍行距
  \doublespacing
- 中文设置
  \usepackagefontspec
  \usepackagexunicode
  \usepackagexeCJK
  设置英文字体
  \setmainfontTimes New Roman
  设置中文字体
  \setCJKmainfont[AutoFakeBold]SimSun 常规字体: 宋体
  \setCJKsansfontSimHei 加粗字体：黑体
  \setCJKmonofontFangSong 等宽字体：仿宋

1.2. 一些有助于排版的小功能（持续更新）

- \noindet 取消段落前的缩进
- \smallskip：插入一个小的空白，用于需要更小间距的情况。
- \bigskip：插入一个较大的空白，用于需要较大间距的情况。
- \medskip 是一个插入适中高度空白的命令，常用于分隔文本段落或元素，增强文档的可读性和排版美感。
- \\ 换行
- \textbf 加粗表示
- \and 表示并列
- \ref 表示引用
- 正在使用的分点表示:
  \begin{itemize}
  \item 分点内容
  \end{itemize}

2. 最短路径算法

2.1. 搜索

搜索问题涉及一个智能体 (**agent**)，该智能体会接收初始状态和目标状态，并返回从初始状态到目标状态的解决方案。例如，导航应用程序就是一个典型的搜索过程，其中智能体（程序的思考部分）接收当前的位置和目标目的地作为输入，并基于搜索算法返回一条推荐路径。然而，搜索问题的形式还有很多其他类型，比如拼图或迷宫等。

搜索过程主要包括以下几个步骤：

- **智能体 (Agent)**:
  一个感知其环境并对环境做出反应的**实体**。例如，在导航应用程序中，智能体可以看作是代表汽车的一个模型，负责决定采取哪些行动以到达目的地。
- **状态 (State)**:
  智能体在环境中的一种**配置**。例如，在 15 拼图游戏中，状态是指所有数字在棋盘上排列的某种方式。
- **行动 (Action)**:
  在某个状态下可以做出的选择。更准确地说，行动可以被定义为一个**函数**。当接收状态作为输入时，返回可以在该状态下执行的**行动集合**。例如，在 15 拼图游戏中，给定状态下的行动是指在当前配置中可以滑动的方块（如果空白方块在中间，则可以滑动 4 个方块；如果在边缘，则是 3 个；如果在角落，则是 2 个）。

- **转换模型 (Transition Model)**:
  描述执行任何适用行动后，所得到的新状态。更准确地说，转换模型可以定义为一个函数。当接收状态和行动作为输入时，返回执行该行动后得到的**新状态**。例如，在 15 拼图游戏中，给定某个配置（状态），移动方块（行动）会导致拼图的新配置（新状态）。
- **状态空间 (State Space)**:
  从初始状态通过任何一系列行动可以到达的所有状态的**集合**。例如，在 15 拼图中，状态空间包含所有可以从任意初始状态通过一系列合法行动达到的 $\frac{16!}{2}$ 种棋盘配置。状态空间可以被看作是一个有向图，**其中状态是节点，行动是节点之间的箭头。**

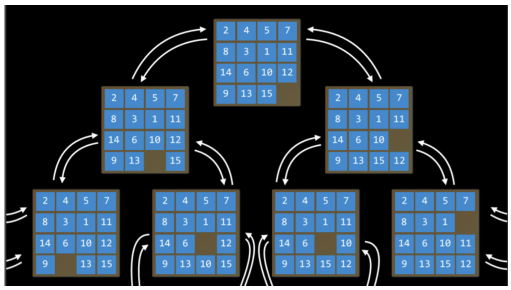

Figure 1. 状态空间

2.2. DFS 与 BFS

- **深度优先算法（DFS）**:
  深度优先搜索算法在尝试另一个方向之前会穷尽每个方向。在这些情况下 **frontier** 作为**堆栈数据结构**进行管理。需要记住的标语是**“后进先出”**。将节点添加到边界后，要删除并考虑的第一个节点是要添加的最后一个节点。**这导致搜索算法在第一个方向上尽可能深入，而将所有其他方向留给以后**。
  一个例子：以你正在寻找钥匙的情况为例。在深度优先的搜索方法中，如果选择从裤子开始搜索，你首先会检查每个口袋，清空每个口袋并仔细检查内容。只有当你完全用尽了裤子每个口袋的搜索后，你才会停止在裤子里搜索，并开始在其他地方搜索。
  - **优点:** 此算法充其量是最快的。如果它“运气好”并且总是（偶然）选择正确的解决方案路径，那么深度优先搜索需要尽可能短的时间来找到解决方案。
  - **缺点：**找到的解决方案可能不是最优的。在最坏的情况下，此算法将在找到解决方案之前探索所有可能的路径，从而在获得解决方案之前花费尽可能长的时间。
- **深度优先算法（BFS）**:
  广度优先搜索算法将同时遵循多个方向，在每个可能的方向上迈出一步，然后在每个方向上迈出第二步。在这种情况下，frontier 作为**队列数据结构**进行管理。需要记住的标语是**“先进先出”**。在这种情况下，所有新节点都会按顺序加起来，并且根据先添加的节点来考虑节点 **先到先得！这会导致搜索算法在每个可能的方向上采取一步，然后再在任何一个方向上采取第二步**。
  一个例子：假设正在寻找钥匙。在这种情况下，如果你从裤子开始，你会看看你的右口袋。在此之后，你将查看一个抽屉，而不是查看您的左口袋。然后在桌子上。以此类推，在你能想到的每个位置。只有在你用完所有位置后，你才会回到你的裤子里，在下一个口袋里搜索。
  - **优点:** 此算法保证找到最佳解决方案。
  - **缺点:** 几乎可以保证此算法的运行时间比最短时间长。在最坏的情况下，此算法需要尽可能长的运行时间。

2.3. 贪心算法和 A* 算法

广度优先和深度优先都是**不知情**的搜索算法。也就是说，这些算法没有利用他们没有通过自己的探索获得的有关问题的任何知识。但是，大多数情况下，实际上可以获得有关该问题的一些知识。例如，当人类迷宫求解器进入一个路口时，人类可以看到哪条路沿着解

决方案的大致方向走，哪条路不走。(总是选择尽量靠近出口的方向选择) 一种考虑额外知识以尝试提高其性能的算法称为**知情搜索算法**。

- **贪心算法**:
  贪婪最佳优先搜索会扩展最接近目标的节点，由**启发式函数** $h(n)$ 确定。顾名思义，该函数估计下一个节点离目标有多近，但可能会出错。贪婪最佳优先算法的效率取决于启发式函数的好坏。
  例如，在迷宫中，算法可以使用启发式函数，该函数依赖于可能节点与迷宫末端之间的**曼哈顿距离**。曼哈顿距离忽略墙壁，并计算从一个位置到目标位置需要向上、向下或向两侧走多少步。这是一个简单的估计值，可以根据当前位置和目标位置的 (x, y) 坐标得出。

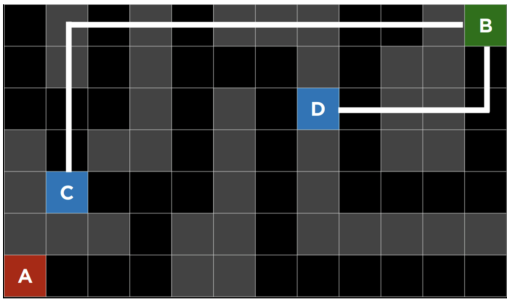

Figure 2. 曼哈顿距离

- **A* 算法**:
  作为贪婪最佳优先算法的发展，$A*$ 搜索不仅考虑 $h(n)$（从当前位置到目标的估计成本），还考虑 $g(n)$（直到当前位置为止的**累积成本**）。通过组合这两个值，该算法可以更准确地确定解决方案的成本并随时随地优化其选择。该算法会跟踪（到目前为止的**路径成本 + 达到目标的估计成本**），一旦它超过前一个选项的估计成本，算法将放弃当前路径并返回上一个选项，从而防止自身走上一条 $h(n)$ 错误标记为最佳路径的漫长、低效的路径。
  再一次，由于此算法也依赖于启发式方法，因此它与它采用的启发式方法一样好。在某些情况下，它可能不如贪婪的最好优先搜索甚至不知情的算法有效。要使 A* 搜索最优，启发式函数 $h(n)$ 应为：
  – **可接受**，或从不高估真实成本，以及
  – **一致**，这意味着除了从前一个节点过渡到新节点的目标的成本外，到新节点目标的估计路径成本大于或等于到前一个节点目标的估计路径成本。用方程式来说，如果对于每个节点 n 和后继节点 n'，步长成本为 c，$h(n) \leq h(n') + c$, 则 $h(n)$ 是一致的。

3. LU 分解

LU 分解是指矩阵 A 可以分解为 LU 乘积的形式，其中 L 是单位下三角矩阵，U 是单位上三角矩阵。
实际上就是用 LU 矩阵保存下来高斯消元的行列线性变换，利于求解 $Ax = b$, 在 b 不确定时将计算从 n^3 化简成 n^2

3.1. textbfGuass 变换

若想将给定矩阵 A 分解为下三角矩阵 L 和上三角矩阵 U，一个思路就是通过一系列的初等变换将 A 化为上三角矩阵，且保证这些变换的乘积是一个下三角，比如通过初等变换 $L_{n-1}L_{n-2}...L_1A = U$，则 $A = L_1^{-1}L_2^{-1}...L_{n-1}^{-1}U$，其中 U 是一个上三角矩阵，$(L_1L_2...Ln)^{-1}$ 是一个下三角矩阵。所以问题就转化为找满足条件的下三角矩阵，对于任意给定的向量 $x \in R^n$，找一个简单的下三角矩阵 L_k 使 x 经过这一矩阵的作用之后的第 k+1 至第 n 个分量均为 0。能够完成这一条件的最简单的下三角矩阵如下：

$$L^k = I - l_k e_k^T \tag{1}$$

$$l_k = (0 \dots 0, l_{k+1,k} \dots l_{nk})^T \tag{2}$$

$$L_k = \begin{pmatrix} 1 & \ddots & & \\ & 1 & & \\ & -l_{k+1,k} & 1 & \ddots \\ & \vdots & & \\ & -l_{n,k} & & 1 \end{pmatrix} \tag{3}$$

3.2. 高斯消元法实现 LU 分解

对于任意给定的向量 $x = (x_1, \dots, x_n)^T \in R^n$，则

$$L_k x = (x_1, \dots, x_k, x_{k+1} - x_k l_{k+1,k}, \dots, x_n - x_k l_{n,k})$$

若要使得第 $k+1$ 至第 n 个分量均为 0，则

$$l_{ik} = \frac{x_i}{x_k}, \quad i = k+1, \dots, n$$

此时

$$L_k x = (x_1, \dots, x_k, 0, \dots, 0)$$

且 Gauss 变换 L_k 的性质非常好，Gauss 变换的逆特别容易求：

$$(I - l_k e_k^T)(I + l_k e_k^T) = I - l_k e_k^T l_k e_k^T = I$$

即

$$L_k^{-1} = I + l_k e_k^T$$

此时

$$A = L_1^{-1} L_2^{-1} \cdots L_n^{-1} U = LU$$

其中

$$L = L_1^{-1} L_2^{-1} \cdots L_n^{-1} = (I+l_1 e_1^T)(I+l_2 e_2^T) \cdots (I+l_{n-1} e_{n-1}^T)$$

$$= I + l_1 e_1^T + l_2 e_2^T + \cdots + l_{n-1} e_{n-1}^T$$

则 L 具有如下形式：

$$L = I+(l_1, l_2, \ldots, l_{n-1}, 0) = \begin{pmatrix} 1 & & & & \\ l_{21} & 1 & & & \\ l_{31} & l_{32} & 1 & & \\ \vdots & \vdots & \vdots & \ddots & \\ l_{n1} & l_{n2} & l_{n3} & \cdots & 1 \end{pmatrix}$$

所以 L 是一个单位下三角矩阵，矩阵 A 可以分解为单位下三角矩阵 L 和上三角矩阵 U 的形式。

3.3. 用法

知道了 L 和 U, 式子 $Ax = b$ 可以写作 $LUx = b$, 写作 $Ly = b$, 先解出 y 这个过程称为**正向迭代**然后再由 $Ux = y$ 解出 x，此过程称之为**反向迭代**。如此就不需要在每次在 b 发生变化时多次进行重复的高斯消元，简化计算机计算量

4. 总结

1 最短路径算法虽然从迷宫的角度来理解还是较为直观和轻松，但是在代码层面实现起来还是需要一定时间去理解。特别是当这类算法抽象用来解决一些表面上看起来不是很直观的问题上面。还得多多学习

2 关于矩阵的计算问题以后也是会经常用的，在搭建网络的时候很有用处。比如课上老师所展示的谷歌的网页排序的问题。就可以用概率矩阵和最初始的停留在网页链接上的人数推断出最终达到稳态时网页停留人数排名

3 这周因为临近的考试，还有学习 latex 的时间，团课啥啥的，导致学的东西除了课内的知识外并不是特别多还是得多多抽出时间学习课外内容，把效率提高。