盲点钻石矿工

Myriad Dreamin 2017211279 2017211301

目录

1 约定

1.1 mp,n,m

本文出现的 mp，总是四连通图模型。n 表示 mp 的行数，m 表示 mp 的列数。

1.2 dp

对于贪心法的 dp, 有二维. 第一维表示行号，第二维保存列号。

1.3 k 阶子式

矩阵 A, 第 $i_1, i_2, \ldots, i_k$ 行, 第 $j_1, j_2, \ldots, j_k$ 列交叉形成的子式的表示法为:

$$A\begin{pmatrix} i_1; i_2; \ldots; i_k \\ j_1; j_2; \ldots; j_k \end{pmatrix}$$

2 程序控制

本次算法测试在 Windows10 平台,c++ 规范为 g++/c++11.

本次算法测试框架为上次排序算法的测试框架的抽象化，接受一个测试匿名函数和断言匿名函数. 框架会在进入测试函数的时候计时，在离开测试函数的时候停止计时。如果定义了宏信号 DEBUG，并且传入了断言函数，框架会调用断言函数.

算法时间测试的粒度为 $1\mu s$.

压入匿名函数的效率很高，并且是强制无参匿名函数，所以压函数栈和退函数栈的时间短于 $1\mu s$，可以将这段时间忽略不计.

此次算法的主要矛盾在于，如何获得最优值，而不是算法效率，因此算法时间复杂度在可以接受的范围内即可。本文也几乎只在意最终获取的总价值。

考虑降低算法输入的规模. 如果地图过大，显然可以将多个格点合并为一个格点，则降低了输入规模。

3 c++ 生成二维正态分布

因为带有 ρ 参数的二维正态分布的难度和代价比较高, 所以本文只使用 $\vec{X} = (x_1, x_2)$ 两变量独立的正态分布.

`functory_range(`σ_l`,`σ_r`)` 接受两个方差值, 返回一个调制函数 `functory`, 这个函数返回一个函数, 函数中包含的两个正态的方差 $\sigma \in [\sigma_l, \sigma_r]$.

`functory` 接受一个二维点 $\vec{\mu} = (\mu_1, \mu_2)$，返回一个表达 (μ_1, μ_2) 服从正态分布的随机向量的函数 `generator`.

`generator` 不接受输入, 每次调用返回一个总体服从在 (μ_1,μ_2) 上各自服从 (σ_l,σ_r) 的向量.

4 已知 $\{\text{Manhattan}(\vec{x}, \vec{y}) \leqslant K\}$ 分布

4.1 贪心法

贪心法原理：

根据动态规划的递推公式。当视野为 K 时，对于当前格点 $F = (a, b)$,Manhattan 距离 $d < K$ 的格点是已知的，显然我们有：

$$\text{dp}[x][y] = \max \text{dp}[x][y+1], \text{dp}[x+1][y] + \text{mp}[x][y].$$

其中 $\text{mp}[x][y]$ 表示真实矿产分布.

如果 $\text{dp}[a+1][b] > \text{dp}[a][b]$，说明往下 (往行号增大方向) 走，在当前视野来看，是更优的。否则说明往右 (往列号增大方向) 走更优。

特殊情况的处理：

1 如果 $a = n, b \neq m$，说明已经走到地图最下侧，此时不能往下走.

2 如果 $b = m, a \neq n$, 说明已经走到地图最右侧，此时不能往右走.

3 如果 $a = n, b = m$，说明已经结束，此时退出算法.

4 如果 $k = 1$, 说明只有当前所在位置的视野，无法进行贪心.

4.2 猜图策略

如何根据已有信息，猜测图剩下的矿产分布。这里有一个很简单的边缘衰减模糊化的方法。

假设矿工现在已经处于 (a,b), 那么:

$$\text{newmp}[a'][b'] = \begin{cases} \text{mp}[a'][b'] & ,\text{Manhattan}((a,b),(a',b')) \leqslant K; \\ c \cdot (\text{newmp}[a'-1][b'] + \text{newmp}[a'][b'-1]), & ,\text{Manhattan}((a,b),(a',b')) > K. \end{cases}$$

4.2.1 c 的选取

对于 (a,b) 的真实值为 d, 它对 $(a+1,b+1)$ 的贡献为 $c(cd+cd) = 2c^2d < d$. 所以 $0 \leqslant c \leqslant \dfrac{\sqrt{2}}{2}$. 否则会使衰减失败，对地图估计错误

4.2.2 特点

1 当 c=0 时，模糊化消失，此时模糊化的贪心原则退化为原始贪心原则.

2 显然这种方法只能估计已经接触到的矿产风貌, 对于完全隐藏在视野外的矿产，模糊化方式估计完全失效。

k	pure greedy	0	0.05	0.1	0.15	0.2	0.25
0	0.429524	0.430476	0.429524	0.474286	0.425714	0.425714	0.442857
20	0.88	0.527619	0.527619	0.527619	0.527619	0.527619	0.527619
40	0.93619	0.651429	0.651429	0.651429	0.651429	0.651429	0.654286
60	0.969524	0.775238	0.775238	0.775238	0.775238	0.775238	0.775238
80	0.958095	0.819048	0.819048	0.819048	0.819048	0.819048	0.820952
100	0.980952	0.980952	0.980952	0.980952	0.980952	0.980952	0.980952
120	0.979048	0.979048	0.979048	0.979048	0.979048	0.979048	0.979048
140	0.982857	0.982857	0.982857	0.982857	0.982857	0.982857	0.982857
160	1	1	1	1	1	1	0.997143
180	1	1	1	1	1	1	1
200	1	1	1	1	1	1	1

表 1: 贪心 K 步和猜图迭代, 部分 1

k	0.3	0.35	0.4	0.45	0.5	0.55	0.6
0	0.40381	0.446667	0.444762	0.408571	0.422857	0.426667	0.417143
20	0.527619	0.522857	0.522857	0.522857	0.51619	0.51619	0.51619
40	0.654286	0.657143	0.657143	0.657143	0.627619	0.627619	0.627619
60	0.775238	0.782857	0.782857	0.782857	0.766667	0.766667	0.752381
80	0.820952	0.819048	0.819048	0.819048	0.822857	0.822857	0.822857
100	0.980952	0.980952	0.980952	0.980952	0.957143	0.957143	0.957143
120	0.979048	0.979048	0.979048	0.979048	0.985714	0.985714	0.985714
140	0.982857	0.99619	0.99619	0.99619	0.991429	0.991429	0.991429
160	0.997143	0.997143	0.997143	0.997143	0.998095	0.998095	0.998095
180	1	1	1	1	0.999048	0.999048	0.999048
200	1	1	1	1	1	1	1

表 2: 贪心 K 步和猜图迭代, 部分 2

5 缺失 c% 分布

随机 c% 个像素已知矿产含量，未知的地方填零。问该地图我们能得到多少价值？

5.1 朴素情况

不作任何处理，直接进行 dp. 这个方法已经较好，此时在 $90\% \sim 95\%$ 以内的未知矿产就能获得很大价值。

5.2 图像推断的思想

通过已知部分的图像还原整个图像信息，是图像补全的思想。现在已经是一个 AI 的热门领域，并且有过大量研究。当图像信息过少，补全难以做到，这个解决方案进一步变为图像推断。但是个人能力有限，从事的领域也不属于 AI 和计算领域，所以这里的图像推断采用另一种原始的方法——卷积模糊化做部分推断。

5.2.1 卷积方式

下文介绍的都是 $3x3$ 卷积核，没有使用其他库，而是做了一个的简单版本使用。

设需要处理的矩阵为 $A_{m\times n}$, 将其扩充为：

$$E(A) = \begin{pmatrix} M_1 & M_2 & M_3 \\ M_4 & A & M_5 \\ M_6 & M_7 & M_8 \end{pmatrix}$$

这样即使是 A 的边缘元素也能做一般化处理。

设卷积核矩阵函数为 $K_{3\times 3} : \mathcal{K}^3 \to \mathcal{K}$，则处理过后的矩阵即为

$$A'_{m\times n} = \left[K(E(A) \begin{pmatrix} a(i)-1; a(i); a(i)+1 \\ a(j)-1; a(j); a(j)+1 \end{pmatrix}) \right]_{m\times n}$$

其中 a 将原 A 矩阵的位置映向 $E(A)$ 矩阵中对应的位置.

这里假设 A 矩阵都已经做了零值填充，即:

$$E_0(A) = \begin{pmatrix} 0 & \mathbf{0}^T & 0 \\ \mathbf{0} & A & \mathbf{0} \\ 0 & \mathbf{0}^T & 0 \end{pmatrix}$$

5.2.2 常滤波

常滤波的核如下：

$$\mathrm{Ker}_c = \frac{1}{9}\begin{pmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

这个核没有任何作用，相当于卷积的单位元.

$$K_c = M(2;2).$$

5.2.3 均值滤波

均值滤波的核如下：

$$\mathrm{Ker}_{ave} = \frac{1}{9}\begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix}$$

均值滤波的特性就仅仅是将附近值收集到一起。

$$K_{ave} = \sum_{i,j} \mathrm{Ker}_{ave}(i;j)M(i;j)$$

5.2.4 高斯滤波

高斯滤波的核如下:

$$\mathrm{Ker}_{gauss} = \frac{1}{16}\begin{pmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{pmatrix}$$

高斯滤波对不同的距离的值做了加权，本质还是一种线性滤波。

$$K_{gauss} = \sum_{i,j} \mathrm{Ker}_{gauss}(i;j)M(i;j)$$

5.2.5 中值滤波

中值滤波是一种非线性滤波，返回矩阵九个值的中值元素。

$$K_{median} = \mathrm{Median}\{M(i;j)\}$$

5.3 模糊化不影响判断

下面是 K_{ave} 作用于矩阵以后，dp 与最优答案的比值表。

scalar	K_c	K_{ave}	$1.1K_{ave}$	$1.2K_{ave}$	$1.3K_{ave}$	$1.4K_{ave}$
100	0.882109	0.879075	0.879075	0.879075	0.879075	0.879075
400	0.801967	0.798407	0.798407	0.798407	0.798407	0.798407
900	0.807186	0.811994	0.811994	0.811994	0.812143	0.811994
1600	0.773705	0.774371	0.774371	0.774371	0.774371	0.774371
2500	0.755642	0.768923	0.768923	0.768923	0.768923	0.768923
3600	0.769577	0.753803	0.753803	0.753803	0.753803	0.753803
4900	0.76023	0.790055	0.790055	0.790055	0.790055	0.790055
6400	0.679542	0.723598	0.723598	0.723598	0.723598	0.723598
8100	0.752281	0.738184	0.738184	0.738184	0.738184	0.738184
10000	0.618809	0.675824	0.675824	0.675824	0.675824	0.675824

表 3: $\sigma \in [1, 2]$, c=0.9

$1.5K_{ave}$	$1.6K_{ave}$	$1.7K_{ave}$	$1.8K_{ave}$	$1.9K_{ave}$	$2K_{ave}$
0.879075	0.879075	0.879075	0.879075	0.879075	0.879075
0.798407	0.798407	0.798407	0.798407	0.798407	0.798407
0.812041	0.811994	0.811994	0.812143	0.811994	0.811994
0.774371	0.774371	0.774371	0.774371	0.774371	0.774371
0.768923	0.768923	0.768923	0.768923	0.768923	0.768923
0.753803	0.753803	0.753803	0.753803	0.753803	0.753803
0.790055	0.790055	0.790055	0.790055	0.790055	0.790055
0.723598	0.723598	0.723622	0.723598	0.723598	0.723598
0.738184	0.738184	0.738184	0.738184	0.738184	0.738184
0.675824	0.675824	0.675824	0.675824	0.675824	0.675824

表 4: $\sigma \in [1, 2]$, c=0.9

可以看见随着 K_{ave} 系数增大，改变价值的能力几乎是不变的。

下面是 K_{gauss} 作用于矩阵以后，dp 与与最优答案的比值表。

scalar	K_c	K_{gauss}	$1.1K_{gauss}$	$1.2K_{gauss}$	$1.3K_{gauss}$	$1.4K_{gauss}$
100	0.821555	0.810076	0.810803	0.811473	0.811373	0.810789
400	0.838839	0.83473	0.834852	0.834639	0.830473	0.830635
900	0.87591	0.854756	0.853718	0.854779	0.852307	0.852081
1600	0.850195	0.822489	0.824201	0.824552	0.824382	0.824137
2500	0.873134	0.854688	0.853763	0.854771	0.854688	0.854851
3600	0.834856	0.819559	0.82257	0.81746	0.82212	0.819933
4900	0.843879	0.829675	0.839065	0.834846	0.835755	0.835437
6400	0.850699	0.845951	0.847727	0.838449	0.844552	0.83858
8100	0.856941	0.846637	0.851155	0.850494	0.850534	0.845772
10000	0.841469	0.836623	0.831355	0.836474	0.835293	0.831937

表 5: $\sigma \in [1, 20]$, c=0.9

$1.5K_{guass}$	$1.6K_{guass}$	$1.7K_{guass}$	$1.8K_{guass}$	$1.9K_{guass}$	$2K_{guass}$
0.810204	0.811473	0.811473	0.811359	0.811473	0.810147
0.834363	0.83597	0.834483	0.83597	0.834483	0.834532
0.853551	0.853543	0.853468	0.85375	0.853543	0.853217
0.817305	0.822378	0.823687	0.818264	0.822769	0.815201
0.854691	0.85444	0.854164	0.85451	0.854562	0.854475
0.816711	0.820401	0.817717	0.816747	0.817361	0.817308
0.835236	0.832567	0.834334	0.830717	0.830232	0.831982
0.837357	0.842895	0.836774	0.839475	0.837098	0.837353
0.840543	0.848036	0.843006	0.841083	0.841248	0.842261
0.831827	0.831899	0.83209	0.829435	0.831691	0.831988

表 6: $\sigma \in [1, 20]$, c=0.9

其与 K_{ave} 相同，对系数均不敏感。

5.4 价值集中程度对滤波估计的影响

下表对比了 $K_{ave}, K_{gauss}, K_{median}$ 受矿产分布方差的影响.

s	K_c	K_{ave}	$1.5K_{ave}$	K_{gauss}	$1.1K_{gauss}$	K_{median}
4	0.901281	0.881119	0.881854	0.889729	0.889788	0.752151
8	0.865934	0.850433	0.854533	0.855052	0.857045	0.865358
12	0.846973	0.832431	0.833705	0.835887	0.835949	0.851627
16	0.831654	0.792379	0.794325	0.809303	0.810575	0.935816
20	0.795934	0.784212	0.774906	0.795984	0.796733	0.972669
24	0.779348	0.759825	0.759205	0.775849	0.768992	0.984889
28	0.781897	0.770312	0.763409	0.770012	0.767325	0.979859
32	0.757469	0.729658	0.723171	0.751625	0.743589	0.971458
36	0.767488	0.743899	0.730179	0.757515	0.740707	0.959412
40	0.765495	0.770273	0.711194	0.763575	0.741977	0.970068
44	0.773219	0.724417	0.721279	0.742892	0.743284	0.952504
48	0.759132	0.706167	0.698156	0.704895	0.716084	0.961911
52	0.774267	0.730719	0.720127	0.779843	0.749484	0.972503
56	0.732951	0.7661	0.727341	0.733267	0.73627	0.978032
60	0.719893	0.802861	0.708655	0.733163	0.756834	0.950068
64	0.729374	0.77226	0.721904	0.747649	0.744581	0.950277
68	0.713734	0.80889	0.710166	0.772246	0.760032	0.933135
72	0.685013	0.803066	0.739057	0.749636	0.777923	0.927328
76	0.713734	0.875498	0.715316	0.75049	0.753021	0.916619
80	0.709158	0.810222	0.712742	0.811215	0.798478	0.966147
84	0.707331	0.904963	0.690769	0.722336	0.731618	0.935253
88	0.727582	0.829838	0.775196	0.770661	0.754563	0.829838
92	0.716539	0.916478	0.785523	0.809961	0.804908	0.916478
96	0.716928	0.857687	0.719328	0.808415	0.808979	0.857687
100	0.721106	0.94229	0.729751	0.768069	0.764486	0.94229

表 7: scalar $= 40 \times 40$, $\sigma \in [\sqrt{s}, 2\sqrt{s}]$, c=0.9

随着方差增大, 除了 K_{median},K_c, K_{ave}, K_{gauss} 的效率下降.

而 K_{median} 的效率是出人意料的，猜想其效率上升的原因是。当 A 中孤立元素过多，K_{median} 滤去了太多细节，因此获得的价值不如不优化。当 σ 增大时，K_{median} 迅速补全缺失的聚集空隙，起到了还原的效果。

5.5 地图规模对滤波估计的影响

下表对比了 $K_{ave}, K_{gauss}, K_{median}$ 受地图规模的影响.

scalar	K_c	K_{ave}	$1.5K_{ave}$	K_{gauss}	$1.1K_{gauss}$	K_{median}
100	0.868048	0.803378	0.803378	0.815335	0.815335	0.949142
400	0.811835	0.796313	0.796313	0.799471	0.799738	0.884973
900	0.858231	0.832306	0.832306	0.846128	0.846128	0.819574
1600	0.823485	0.807151	0.807151	0.824684	0.824684	0.644509
2500	0.813963	0.777454	0.777428	0.775654	0.775654	0.664381
3600	0.818656	0.786466	0.786466	0.794831	0.794831	0.714038
4900	0.811648	0.803236	0.80318	0.803264	0.803264	0.655265
6400	0.812331	0.793155	0.793155	0.799439	0.799439	0.526234
8100	0.802	0.784369	0.784369	0.782826	0.782826	0.472096
10000	0.81326	0.816825	0.81694	0.822017	0.822017	0.366435

表 8: scalar $= [100, 10000]$, $\sigma = \sqrt{10}$, c=0.9

选取了较为适中的 σ, 在这个环境下, 明确看见，地图规模越大，估计越差。

5.5.1 对 K_{gauss}, K_{ave} 的进一步测试

辛辛苦苦写出了 K_{gauss}, K_{ave}，看见其效果这么差. 心有不甘，但是做过许多测验，最终都没能将这两个滤波器拯救回来. 仔细想想，3x3 核的作用实在有限，看来算法不是一味地生搬硬套。

6 总结

文章第一部分针对 k 步矿工问题提供了两个策略，一个贪心策略，一个猜图策略。贪心策略已经有比较好的表现，而猜图策略能够略微增加价值收益。

文章第二部分针对 $c\%$ 矿工问题测试了基于几个滤波器的猜图策略。对于均值滤波器和高斯滤波器，并未有较好的表现，可能原因是滤波器影响范围太小，对图像补全没有很好的效果。

值得关注的是中值滤波器，它在有限的范围内 (地图规模较小，或地图矿产分布较为分散) 极大地提高了价值获取的效率。但是更加值得注意的一点是，即使有 90% 的地图不知道，K_c 仍然有较好的效果，这时候应该果断根据已有信息取得价值，而不是歪曲信息想方设法完全还原地图。

本文所有代码实现不依赖除 C++ 标准库 Algorithm 中的有限简单函数，和不值得重复实现地无关紧要的函数以外其他的任何库，均有算法实现。