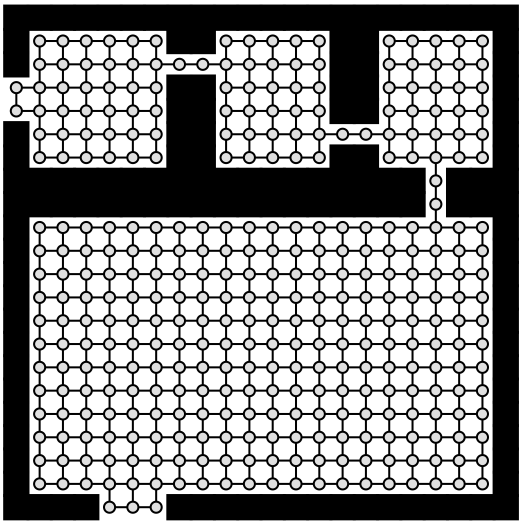
project 10: clustering waypoint graphs

task 10.1: preparation

Download the file `simple-map-dungeon.txt` and use it to build a 2D grid graph $G = (V, E)$ which, when plotted, should look like this:



In lecture 11, we discussed algorithms that can cluster such graphs into k (more or less coherent) sub-graphs and this is what the following tasks will be about.

Note: throughout, we let $n = |V|$ denote the number of vertices of G.

task 10.2: k-medoids clustering

Given G, compute an $n \times n$ distance matrix $\boldsymbol{D}$ whose entries are given by

$$D_{ij} = d(v_i, v_j)$$

where $d(v_i, v_j)$ denotes the length of the shortest path between vertex v_i and vertex v_j.
Once $\boldsymbol{D}$ is available, it should be straightforward to implement Lloyd's algorithm for k-medoids clustering of the vertices of G.

Indeed, those who know how to take advantage of python's `numpy` library for numerical computing will find that it provides functionalities which allow for rather efficient implementations of the following pseudo code

randomly select k initial medoids among the vertices of G

$$\{m_1^1, m_2^1, \ldots, m_k^1\} \subset V$$

for $t = 1, \ldots, t_{\max}$

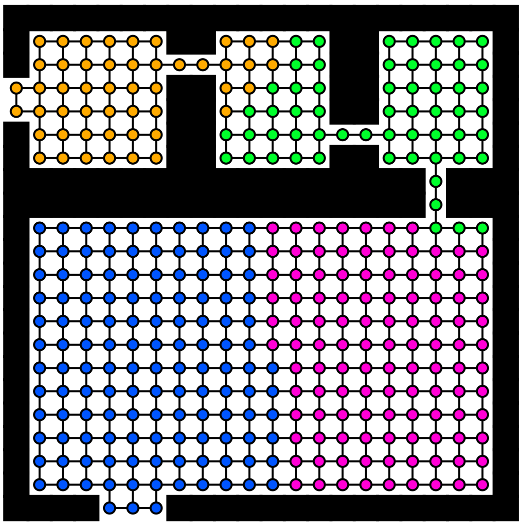
for $i = 1, \ldots, k$ // update clusters

$$C_i^t = \left\{ v_j \in V \;\middle|\; d\big(v_j, m_i^t\big) \leq d\big(v_j, m_l^t\big) \right\}$$

for $i = 1, \ldots, k$ // update medoids

$$m_i^{t+1} = \underset{v_l \in C_i}{\operatorname{argmin}} \sum_{v_j \in C_i} d\big(v_j, v_l\big)$$

Run your program for $k = 4$ and (try to) visualize the clusters you obtain. If you are lucky, your results will look something like this:



Note: if your clusters do not look like this at all, it is always a good idea to diligently check your code for bugs ... However, the most likely reason is the random initialization of the medoids. Hence, run your program several times and ponder the implications of what you observe.

Run your program (several times) with other choices of k and discuss what you observe.

task 10.3: spectral clustering

Given G, compute its $n \times n$ adjacency matrix $\boldsymbol{A}$ where

$$A_{ij} = \begin{cases} 1, & \text{if } (v_i, v_j) \in E \\ 0, & \text{otherwise} \end{cases}$$

Given $\boldsymbol{A}$, it easy to determine the degree $d(v_i)$ of any vertex v_i of G. In fact, convince yourself that

$$d(v_i) = \sum_{j=1}^{n} A_{ij}.$$

Note: the meaning of d has changed. In task 10.2, it referred to a distance between vertices, here it is a vertex degree.

Using the above, compute the $n \times n$ degree matrix $\boldsymbol{D}$ where

$$D_{ij} = \begin{cases} d(v_i), & \text{if } i = j \\ 0, & \text{otherwise} \end{cases}$$

Note: the meaning of $\boldsymbol{D}$ has also changed. In task 10.2, it was a distance matrix, here it is a degree matrix.

Given $\boldsymbol{A}$ and $\boldsymbol{D}$, compute the graph Laplacian

$$\boldsymbol{L} = \boldsymbol{D} - \boldsymbol{A}$$

and then implement the following procedure

1) compute the spectral decomposition of $\boldsymbol{L}$

$$\boldsymbol{L} = \boldsymbol{U} \boldsymbol{\Lambda} \boldsymbol{U}^\intercal$$

(when working with `numpy`, it is strongly suggested to use `numpy.linalg.eigh`)

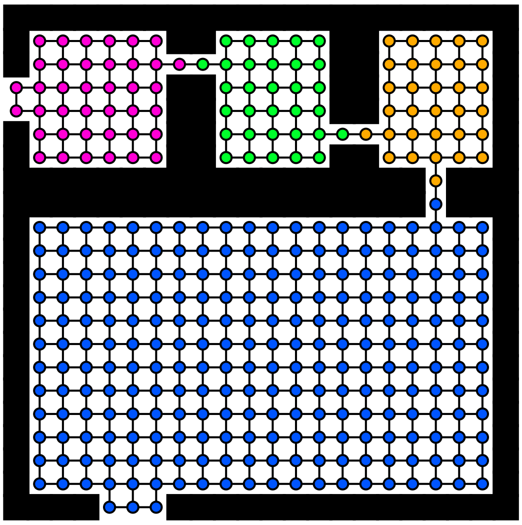
2) assuming the eigenvalues λ_i of $\boldsymbol{L}$ are ordered ascendingly, collect columns $\boldsymbol{u}_2, \boldsymbol{u}_3, \ldots, \boldsymbol{u}_{k+1}$ of $\boldsymbol{U}$ in an $n \times k$ matrix

$$\boldsymbol{W} = \left[\boldsymbol{u}_2, \boldsymbol{u}_3, \ldots, \boldsymbol{u}_{k+1}\right]$$

(observe that i-th row of $\boldsymbol{W}$ constitutes a feature vector for vertex v_i of G)

3) run k-means clustering to cluster the n rows of $\boldsymbol{W}$ into k clusters.

Run your program for $k = 4$ and (try to) visualize the clusters you obtain. (Again, there is a one to one correspondence between the rows of $\boldsymbol{W}$ and the vertices of G). Your result will ideally look like this:



Note: if your clusters do not look like this at all, the most likely reason is the (random) initialization of whatever k-means clustering function you use. Hence, run your program several times and ponder the implications of what you observe.

Note: those familiar with spectral clustering might also try to implement a recursive algorithm that uses the Fiedler vector of G to split G into two sub-graphs and then uses the Fiedler vectors of the sub-graphs to split those into sub-graphs and so on.

Run your program (several times) with other choices of k and discuss what you observe.