#HASH CODE 2017 by Google
Final round

Router Placement

Report

Created by: Yuriy Shtokhman,
Hrytsiv Taras.

#HASH CODE 2017 by Google
Final round

Problem Statement

Given a building plan, decide where to put wireless routers and how to connect them to the fiber backbone to maximize coverage and minimize cost.

Building

The building is represented as a rectangular grid of cells of H rows and W columns. The cells within the grid are referenced using a pair of 0-based coordinates [r, c] , denoting respectively the row and the column of the cell.

Routers

Each router covers a square area of at most $(2 \times R + 1)^2$ cells around it with Internet access, unless the signal is stopped by a wall cell.

Backbone

Routers can be only placed in a cell that is already connected to the fiber backbone (*backbone is a cable that delivers Internet to the router itself*). In the beginning, exactly one cell in the building is connected to the backbone.

Input data

The input data is provided as a data set file - a plain text file containing exclusively ASCII characters with a single “\n” character at the end of each line

The description of each row contains ***W*** characters specifying the type of each cell, one column after another from column 0 to column $W - 1$. Each character is either “#” (denoting a wall cell), “.” (denoting a target cell) or “-” (denoting a void cell).

#HASH CODE 2017 by Google
Final round

Output data

A dictionary with scores and time for each map and overall score, .txt map with routers placed.

Algorithms for placing routers

1. Picky bore

1.1. General idea

This algorithm belongs to the family of algorithms, which use selection, since it goes through the whole map and stores coverage area for each variant of router placement, and then chooses the best one.

1.2. Description

Picky bore algorithm comes through each cell one by one row by row and finds the area, which is covered by the router, which could be placed there. As we get all the routers and areas of coverage we order them from highest to lowest and find the intersection in covered areas. Then the router which covered more space is finally set and any others with the smaller coverage areas are being removed. In such way we get the most optimal position for each router. This algorithm takes a lot of time to be executed, but it’s quite effective and gives great results.

2. Greedy lazybones

2.1. General idea

This algorithm belongs to the family of greedy algorithms, since it chooses the best router (by coverage area) at each step, comparing previous router placement with current.

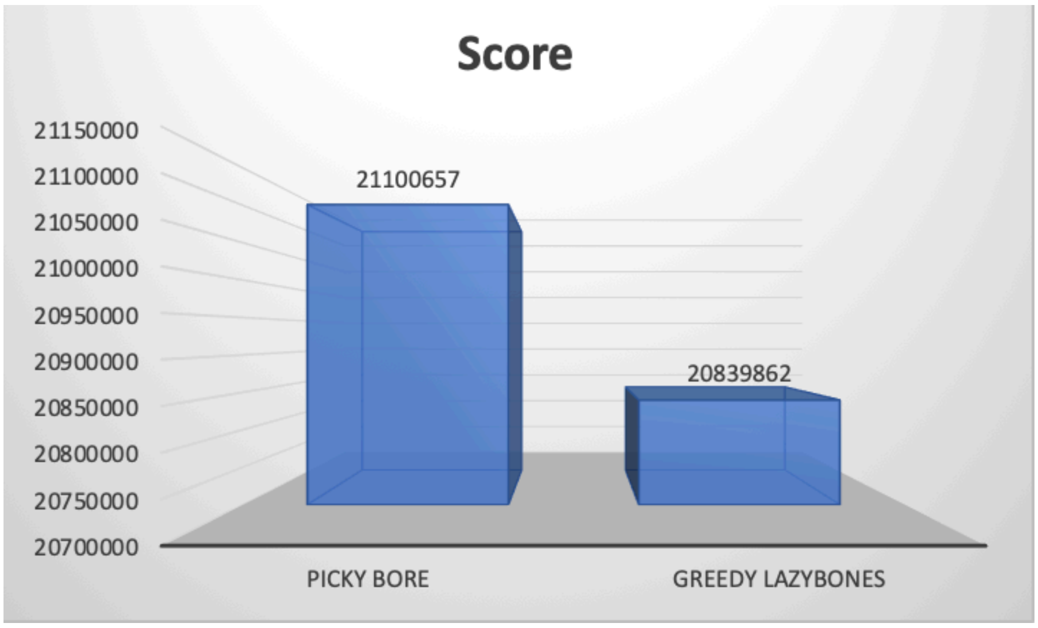
2.2. Description

Greedy lazybones algorithm was developed for minimizing calculation time with ~3% score loss if to compare with *Picky bore* algorithm but strong time benefits (19 sec. for *Charleston road* against 5500 sec if to compare with *Picky bore*). It goes row by row linearly and column by two columns, looking for '.' cell (space which should to be covered) and then goes column by column until radius length not exceeded or #/- symbol found. It helps not to put routers close to another or to wall. Now, when place for router found, it's coordinates and coverage area are being stored into dictionary, next cell to place router is being looked for and compared to previous variant. Best (by coverage area) chosen and covered area marked. Then the dictionary is cleared and next two cells will be found and compared (greediness).

Scores

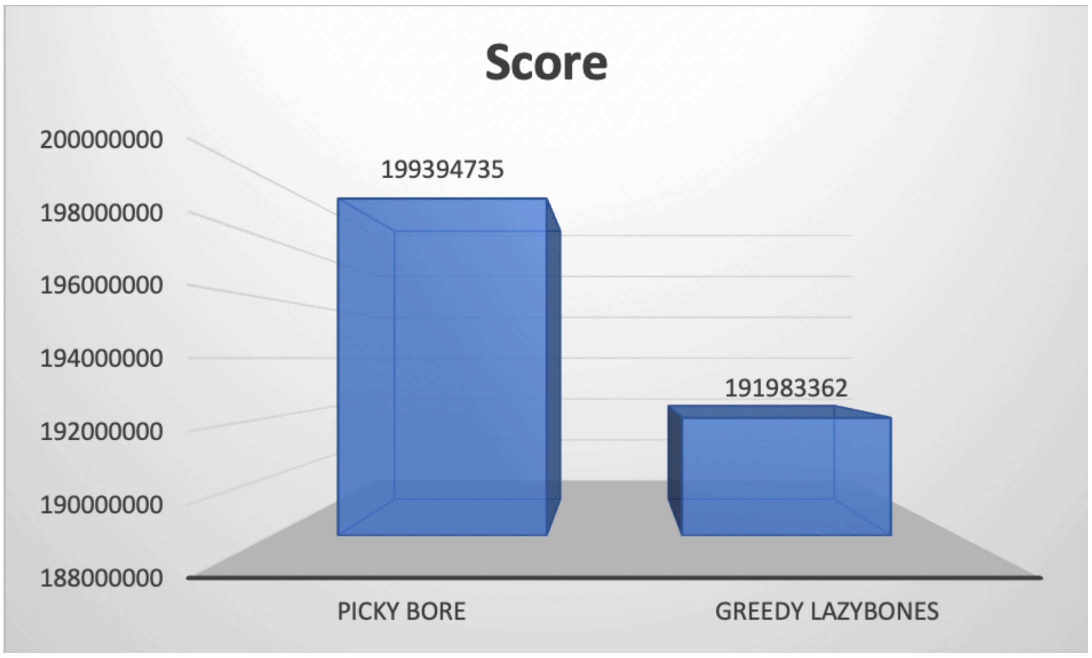
1. The Charleston road

Algorithm	**Score**
Picky bore	21100657
Greedy lazybones	20839862



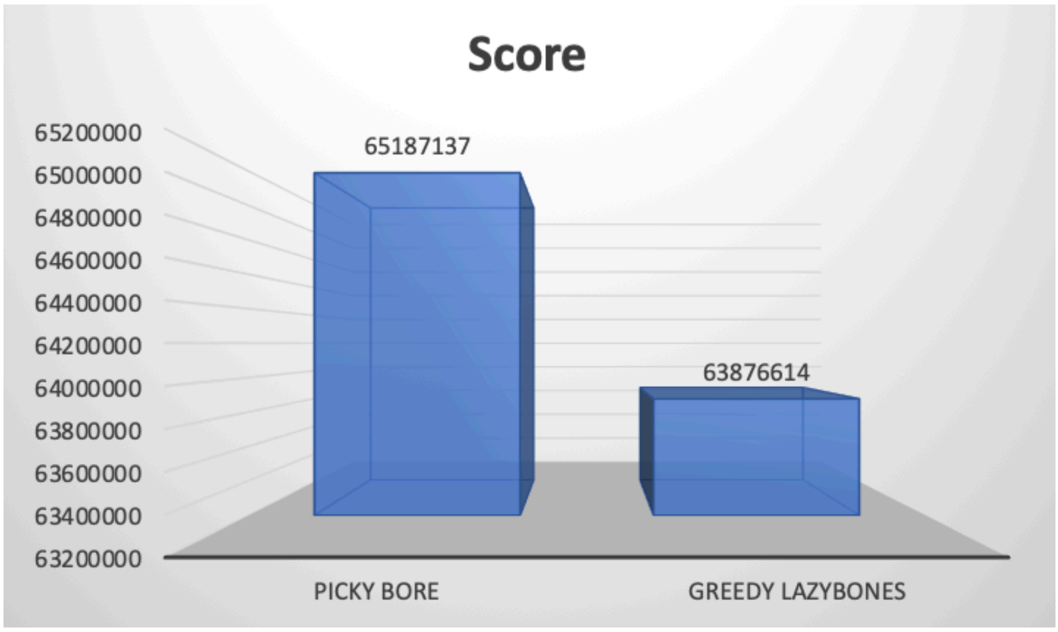
2. Opera

Algorithm	**Score**
Picky bore	199394735
Greedy lazybones	191983362



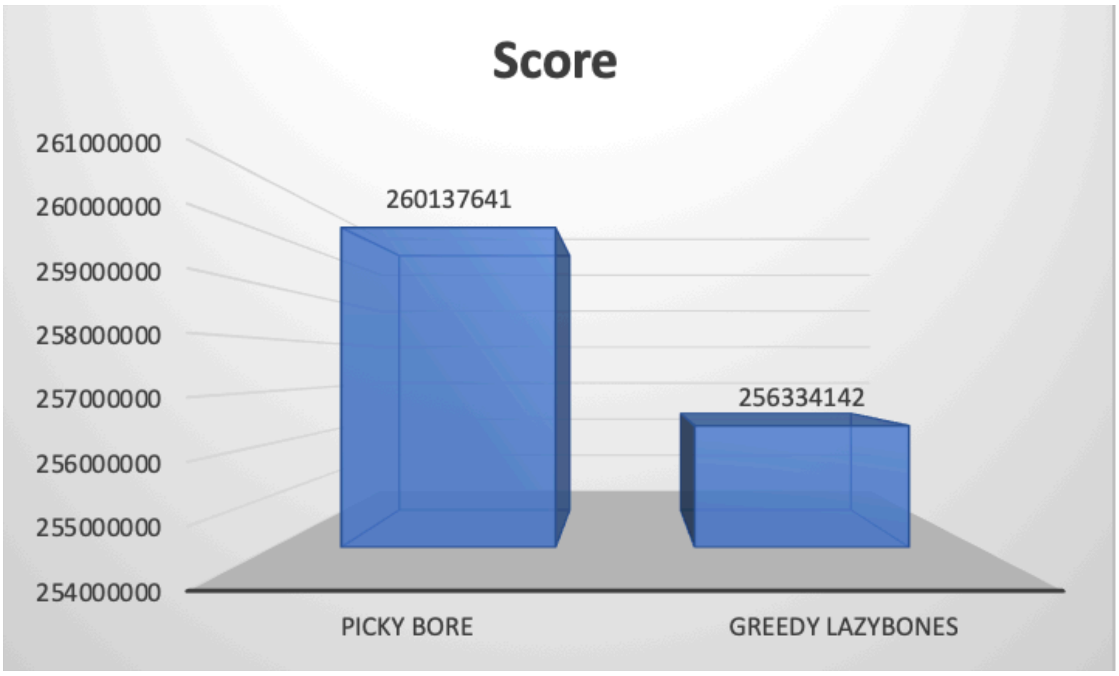
3. Rue de Londres

Algorithm	Score
Picky bore	65187137
Greedy lazybones	63876614



4. Let`s go higher

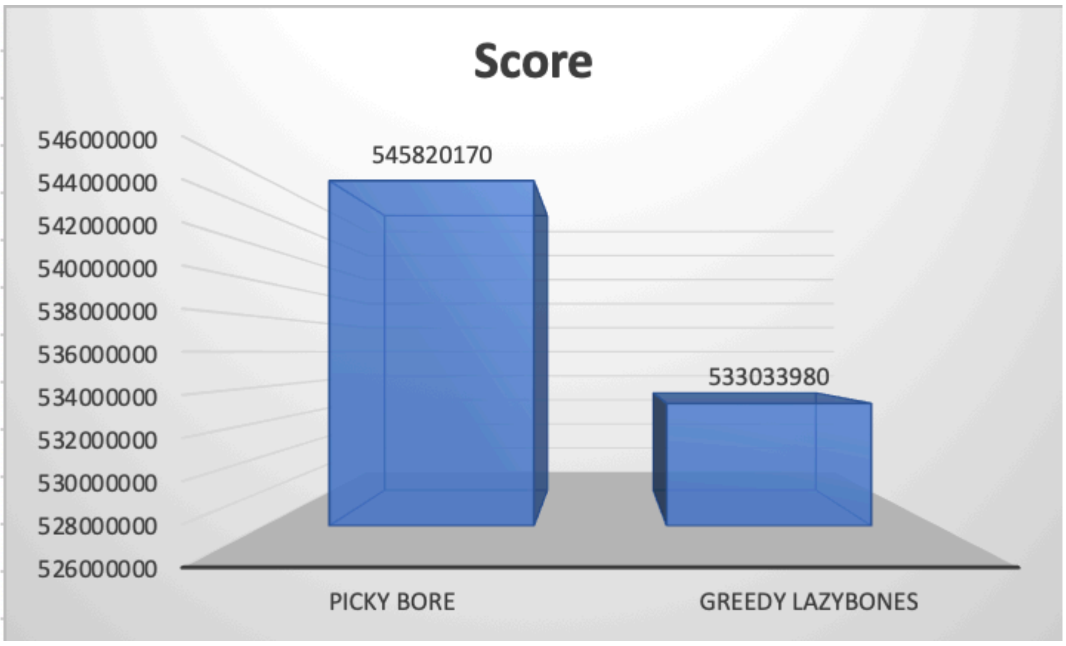
Algorithm	Score
Picky bore	260137641
Greedy lazybones	256334142



#HASH CODE 2017 by Google
Final round

Total score

Algorithm	**Score**
Picky bore	540820170 (#36)
Greedy lazybones	533033980 (#47)



Conclusion

For solving the “*Router placement*” problem our team has developed two algorithms, which where using greedy and selection principles as their base.

Algorithm Picky Bore, due to its strong analysis checks much more options rather then Greedy lazybones algorithm, which makes it better in score results.

However, Greedy lazybones` main feature is speed. It is >250% faster than the first one (score loss can be less in future).

We have a goal to improve even more this algorithms, since if to run Greedy lazybones two times it will give better score, so there still are things to be reconsidered, just like in the Picky bore, where some iterations can be missed, probably.