Minimum Cost Matching in Bipartite Graphs

Group Details:

Ayush Agrawal 14D070042
Mehul Shah 130020090
Pratik Brahma 14D070003
Yash Sanghvi 13D070042

TABLE OF CONTENTS

ABSTRACT

In this project, we take a Boolean function as an input and takes the corresponding gates (AND, OR,XOR, NOT) and place them on a layout grid keeping the variable i.e. inputs at a constant position. The layout is determined using an LP solver given the constraints such as minimum separation and cost per unit length of wire. However a major issue with this approach is that the grid has some discrete points on which we can place the gates. This leads us to 'snapping' problem where the gates are moved towards the discrete points such that the overall length of the wire added is minimum. This problem can be framed as a bipartite graph whose minimum cost matching is required. As a part of the project we solve this problem using the **cycle cancellation algorithm**. To solve the minimum cost bipartite matching problem, we firstly convert the graph into a minimum cost flow problem by forming an equivalent graph, solve the problem through cycle cancellation algorithm and then extracting the edges with non-zero flow. We then declare them as minimum cost edges.

CONSTRUCTING AN EXPRESSION TREE

The user inputs a raw logical expression which must be converted in the form of an expression tree so that we could identify the inputs to the expression and the types and number of gates required to build the expression. The expression tree also establishes the information of the connection between the gates and their hierarchical order. This information of connection between the gates and the inputs is then forwarded to the LP solver to get the desired locations on the board for minimizing the length of wires needed for the connections.

EXPLODED VIEW OF THE ALGORITHM

The algorithm can be divided into three parts:

1. The given infix expression of the user is converted into a postfix expression according to the priority levels defined for the logical expressions.
2. The postfix expression obtained from above is then converted into an expression tree to obtain the connection between the gates.
3. Information is extracted from the expression tree to extract fanin data (one form of representation of a graph) and also assign the input variables as fixed cells.

Operations and their Priorities

Our program supports 5 operations (or, nor, and, xor and nand) whose priority is as defined below

Operation	Symbol	Priority Level

Or	+	0
Nor	$	0
And	*	1
Xor	^	1
Nand	#	1
Not	!	2

Explaining the Operation using an Example

User input Logical Expression : $\boldsymbol{a + (b * c\text{^}d + e) * f}$

1. **Conversion to Postfix Expression**
   The resultant Postfix Expression is : $\boldsymbol{abc * d\text{^}e + f * +}$ **.** The expression is formed according to the above priority
2. **Conversion of Postfix to Expression Tree**
   Below is the image obtained of the expression tree for the above expression

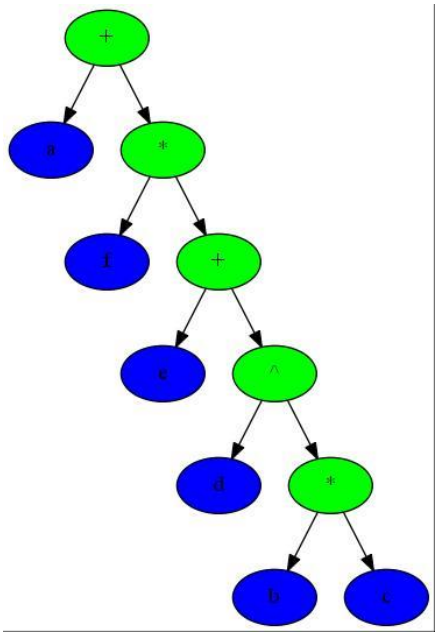


Figure 1. Obtained Expression tree

3. **Extraction of information from the Expression Tree**

   From the expression tree the data obtained are

Fanin	[[], [], [], [], [], [], [2, 1], [3, 6], [4, 7], [5, 8], [9, 0]]
Fixed Cells	[0, 1, 2, 3, 4, 5]

Fanin: A form of representation of the graph
Fixed Cells: Nodes of input variable cells

TWO-DIMENSIONAL LP SOLVER

After getting the expression tree and the board parameters (on which the placement should be done) we must find the optimum positions where each node should be placed. This task is done by the 2-D LP solver. We start by constructing some matrices which satisfy some constraints and try to minimize the total length of the wire (or routing) used. The output of this section of the code is a cost list which provides us with the costs (i.e. distances) from each node (in the optimal solution) to every point on the

board, on which a node can be snapped into. The output also includes a list of edges between each node in the new graph.

EXPLODED VIEW OF THE ALGORITHM

The algorithm works in three parts:

1. Constructing the constraint matrix of the type:
$$Ax \leq b$$
2. Solve the above Linear Programming problem using the 'cvoxpt' library inbuilt into the python. As the output of this step we get one of the optimal solutions as the positions of nodes on the board.
3. Next we construct the main bipartite graph, which consists of set of nodes derived from the expression tree and all the set of coordinates on the board to which one of the actual nodes can be snapped to. We also introduce two extra nodes as Source Node and Terminal Node, which are required in the cycle cancellation algorithm. Now we construct the list of all the edges on the new graph and the cost list – corresponding to each edge on the new graph.

Constraints of the Linear programming problem

The following constraints are added to solve the problem:

$$0 \leq x_i \leq \frac{grid_X}{minimum\ seperation} \qquad for\ all\ i$$
$$x_p = X_p \qquad for\ all\ p\ among\ fixedCells$$
$$0 \leq y_i \leq \frac{grid_Y}{minimum\ seperation} \qquad for\ all\ i$$
$$y_p = Y_p \qquad for\ all\ p\ among\ fixed\ cells$$
$$Lx_{ji} \geq x_i - x_j \qquad for\ all\ j\ \in Fanin(i)$$
$$Lx_{ji} \geq x_j - x_i \qquad for\ all\ j\ \in Fanin(i)$$
$$Ly_{ji} \geq y_i - y_j \qquad for\ all\ j\ \in Fanin(i)$$
$$Ly_{ji} \geq y_i - y_j \qquad for\ all\ j\ \in Fanin(i)$$
$$Lx_{ji} \geq minimum\ seperation \qquad for\ all\ j\ \in Fanin(i)$$
$$Ly_{ji} \geq minimum\ seperation \qquad for\ all\ j\ \in Fanin(i)$$
$$a_s = 0.0$$
$$a_t \leq\ r_t$$
$$a_i \geq a_j + delay_i + alpha * \left(Lx_{ji} + Ly_{ji}\right) \qquad for\ all\ j\ \in Fanin(i)$$

Here the meaning of the symbols used is as follows:

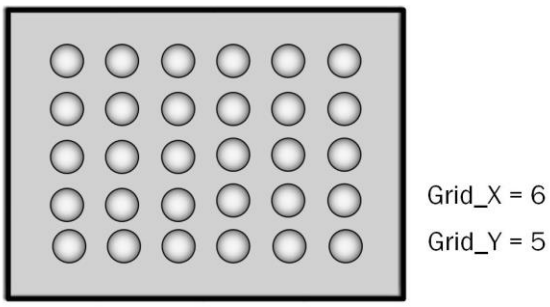
$L \equiv Length\ between\ two\ nodes$
$alpha\ \equiv cost\ for\ unit\ length$
$a\ \equiv AAT\ for\ the\ node$
$(x, y) \equiv coordinates\ of\ the\ node$

The above said constraints ensure that the points lie inside the board boundaries, have some minimum possible distance between them and some gates are present exactly where they must be present.

Construction of the new Bipartite Graph

We get the board's dimensions and the set of points, say N, on which we can snap the nodes from the user input (FIGURE 1). Now for solving the problem we need to make a graph and insert N as nodes in the graph. Now these nodes are put into to a set, say B. Also, the nodes from the expression tree are also introduced into this graph and put into another graph, say A. As already stated above we introduce two extra nodes say S and T for the purpose for solving the problem. Now we add edges to this graph in the following manner (FIGURE 2):

1. S is joined to every node in set A
2. T is joined from every node in set B
3. Every node in set A is connected to every node in set B (These are bipartite matched by the next algorithm)



Example of typical "Board" input given to the code

Figure 1

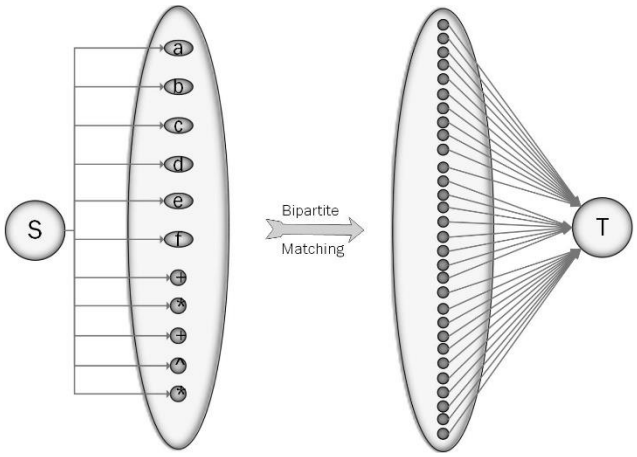


Figure 3

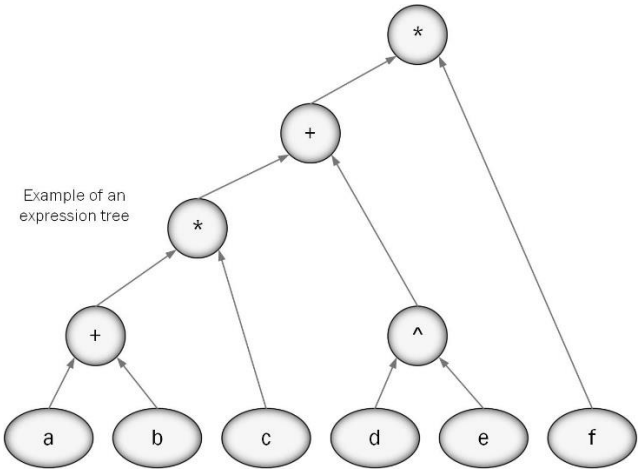


Figure 4: Expression tree for Figure 1 and Figure 2

CYCLE CANCELLATION ALGORITHM

Given a bipartite graph, our objective is to find out a minimum cost perfect matching. This can be done by solving an equivalent minimum cost flow problem and then choosing the edges with non- zero flow. The chosen edges would correspond to the minimum cost matching.

I. Conversion to minimum cost flow problem

Let the given bipartite graph be G= (A ∪ B, E) where A represents the set of nodes on the first connected components, B represents the set of nodes on the second connected components and E represents the set of edges. The edges in E are supposed to satisfy the condition that the outgoing node lies in A and the outgoing node lies in B.
Let the corresponding graph for minimum cost flow problem be G' = (A ∪ B ∪ (source [s], sink [t]), E). Now each edge in the set E has a capacity 1 and cost as the same in graph G. the graphs. Now we also augment edges from source[s] to every node in A and from every node in B to sink[t]. The capacity of all these edges is 1 and the cost is zero.

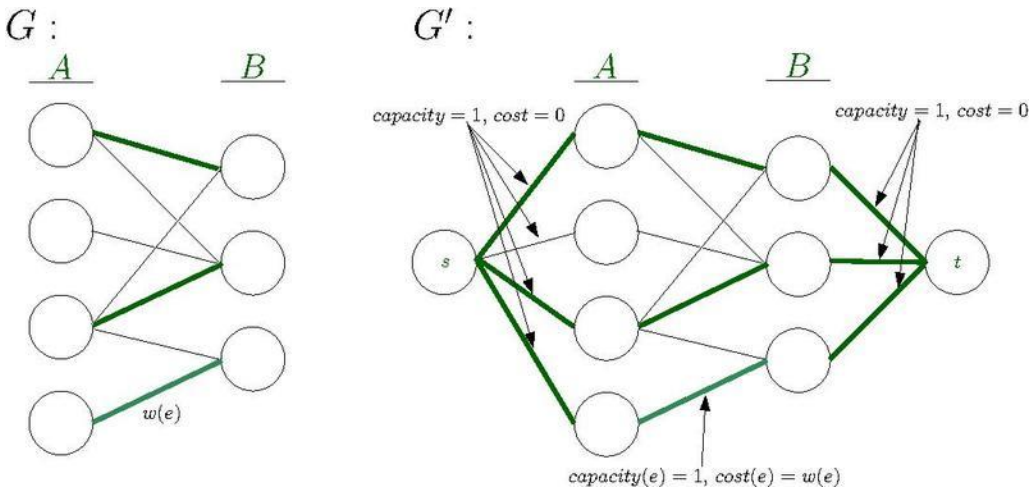


Figure 5 : Minimum cost matching to minimum cost flow problem

II. Cycle Cancellation

Now that we have found the equivalent graph G', we solve the minimum cost flow problem for it.
The objective is given the graph G', a source node 's', a sink node 't' and a given amount of flow 'f', we need to find out the flow through each edge of a graph such that

(i) minimum cost is achieved and
(ii) outflow of source = outflow of sink = f

The whole process can be broken down into two stages. Firstly, we initialize a flow 'f' in the graph using the Edmond Karps algorithm. Note that at this stage, the flow is not optimal in terms of cost.

Secondly, we generate a residual network of the graph. Then we find a negative cost cycle using Edmond Karps algorithm and then find the minimum flow which we can add to this cycle. The residual graph is

updated accordingly and then the second step is repeated until there exists no negative cycle in the graph. This process ultimately gives the optimal flow in terms of cost.

Pseudocode:

Cycle Cancellation (Graph: G, source node: s, sink node: t, flow: f):

- *Establish a feasible flow of 'f' in the network: $Flow_G$*
- *Generate Residual network G_{res}*
- *while (G_{res} has a negative cost cycle):*
  - *Compute the negative Cost cycle C*
  - *Find the minimum flow edge in C = 'd'*
  - *For edges in G and C : add 'd' to the $Flow_G$*
  - *For edges in G_{res} and C: subtract d from the flow (Update to residual network)*
- *Return $Flow_G$*

a. Flow Initialization (Edmond Karps Algorithm)

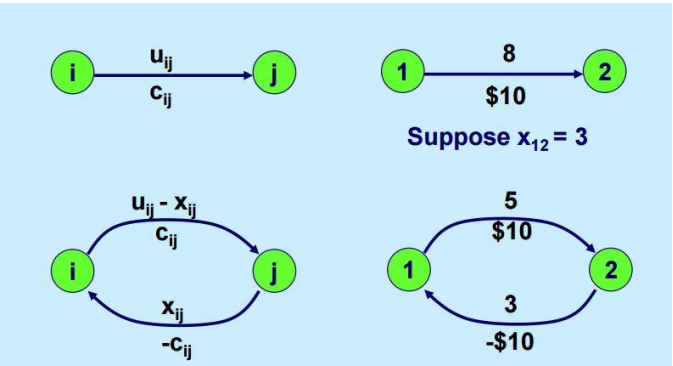
Given f' is the maximum flow in a graph G, the argument as G and flow f, Edmond Karps algorithm returns a feasible flow f in the graph G if f < f' else returns the maximum possible flow f'. Initially the flow is made to be zero and then we keep finding augmenting paths in the residual graph until there are none left and then add the minimum capacity edge's capacity in the path to all edges in the path.

b. Residual Networks

We generate the residual network by the following rule. If there is an edge from V1 to V2, with cost c12, capacity u12 and current flow x12, then the residual graph has an edge

(i) From V1 to V2, with capacity u12 – x12, cost c12

(ii) From V2 to V1, with capacity x12 and cost –c12



Source: MIT OCW Network optimization Fall 2010

Figure 6: How to generate residual graphs

c. Negative Cycles (Bellman Ford)

To compute the negative cycles in the graph we use bellman ford algorithm which is basically used to compute minimum distance path between two vertices. It runs in O(|V||E|) time. The algorithm is described as follows:

Firstly, we obtain a list of distances and predecessors to each node, initialized to infinity and null respectively.

Then for |V| -1 iterations, we 'relax' edges repeatedly i.e. update the distance to a node and its predecessor if the distance to it can be shortened by taking a different edge. Then we check for negative cycles by canning all the edges and by finding a path which has length |V| which can only happen if atleast one negative cycle exists in the graph.

Results:

1. For a working example, we took the following function:

   F(a,b,c,d,e,f,z) = [{(a OR b) OR (c AND e)} XOR (d OR z)] OR f

   The corresponding expression tree is as follows:

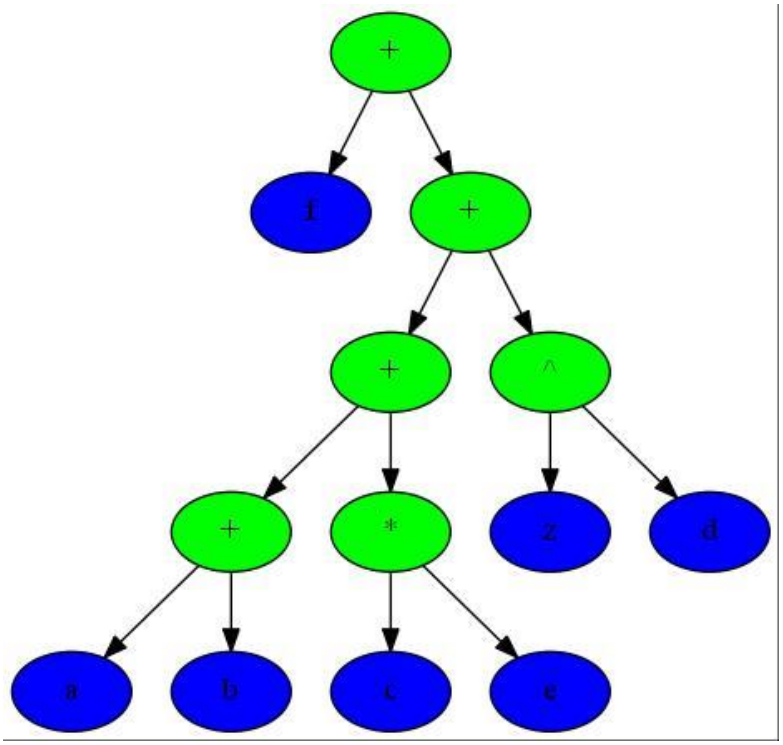


Figure 7: Expression Tree of the Function F(a,b,c,d,e,f,d,z). The green nodes represent an operation and the blue nodes represent a variable.

2. Next, to form the corresponding we allot one node each to a variable and an operation (OR, AND, XOR). The optimal locations (keeping the locations of 'variable' nodes predetermined) of the 'operator nodes' in terms of the length of wire are found out using the Linear Programming Formulation of the problem. The following are the parameters used:
    a. Minimum Separation between two nodes : 0.1
    b. Cost per unit length (Alpha) = 1

    Using these parameters, we obtain the initial position of nodes as follows:

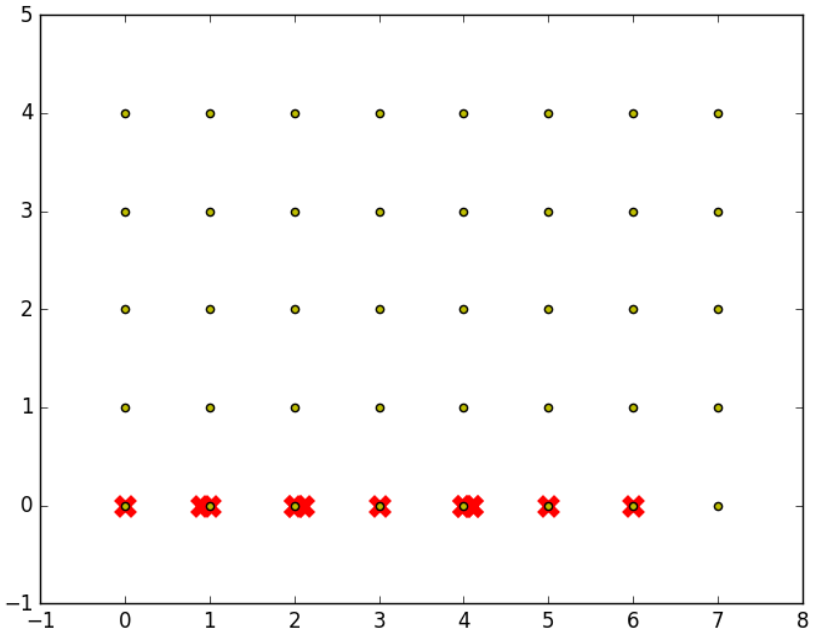


Figure 8: Initial Node positions are represented by a red cross. Yellow dots are the discrete positions available on the graph. Note that all these nodes don't lie on the discrete positions yet.

3. Now we turn the ‘snapping’ of nodes problem into a bipartite weighted minimum cost matching problem. Then we solve it using Cycle Cancellation algorithm and obtain the following discretization of the position of nodes.

```
Node: (0.9, 0.0) snapped to (1, 1)
Node: (2.1, 0.0) snapped to (3, 1)
Node: (2.0, 0.0) snapped to (2, 1)
Node: (4.1, 0.0) snapped to (4, 1)
Node: (4.0, 0.0) snapped to (4, 2)
Node: (4.1, 0.0) snapped to (5, 1)
```

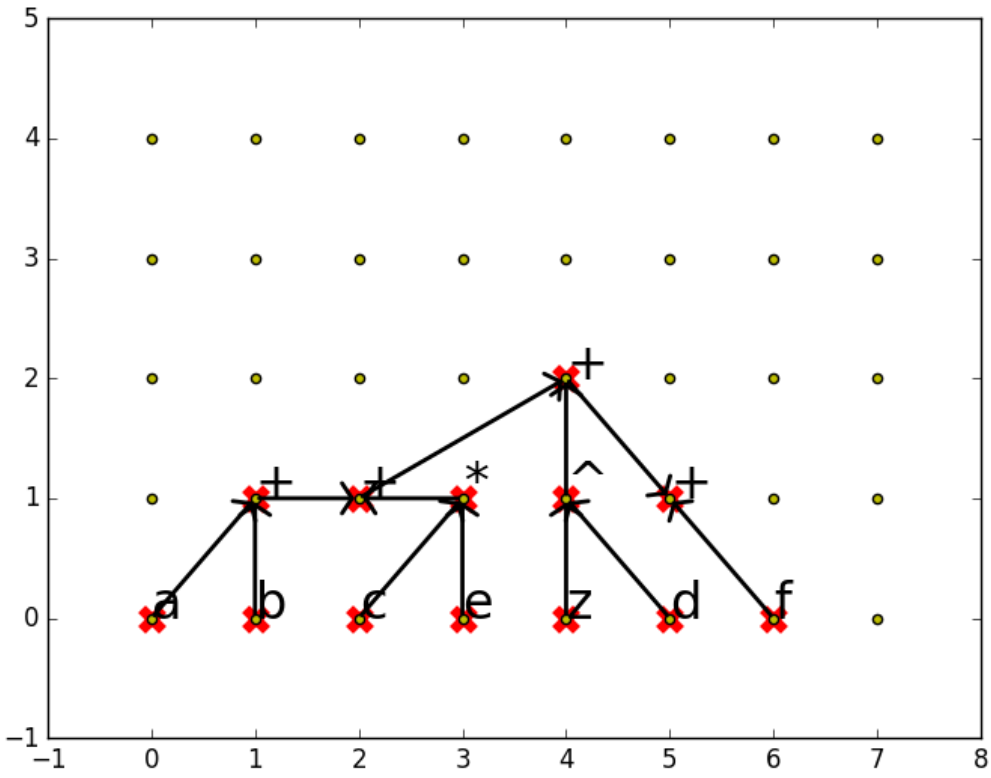


Figure 9: Grid showing discretized locations of each node (represented as a red cross). The arrows represent the fanin and fanout of each node.

FUTURE PROSPECTS

EFFICIENT CITY LAYOUTS

The objective of this problem is to plan the positions of houses in the city and the layout of roads. The positions of the houses should be decided such that the cost of laying the roads connecting among the houses is minimum. The houses can be considered as logical nodes which are to be placed in the board (city dimensions). The roads are connected to main highways which is connected to the city only through some specific positions (input variables). The problem can be explained with an example

An Example

Suppose we have three houses in our small city and three main outlets to the highways form our city. The houses are the operations and the outlets are our variables. Each of the three houses is connected to a single highway and the houses are connected to the adjacent houses.

The expression required :- $\boldsymbol{a} + \boldsymbol{b} + !\,\boldsymbol{c}$

$+, +, !: \boldsymbol{Houses}$

$\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c} - \boldsymbol{Outlets\ to\ Highways}$

The required expression tree is given as below

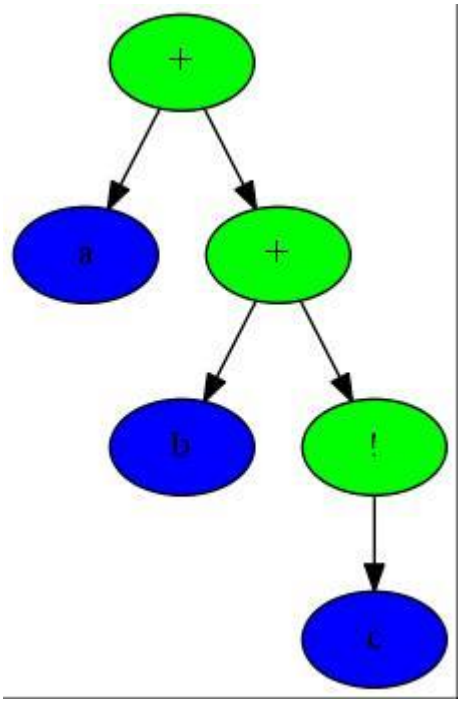


Figure 10

The required layout of houses and roads

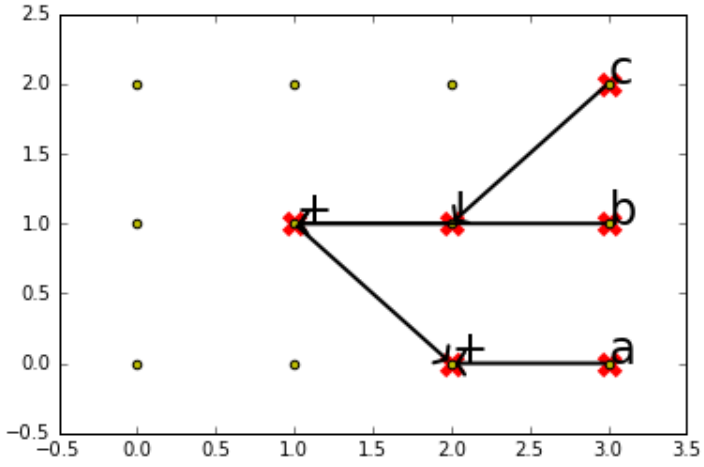


Figure 11

The above figure 11 represents the layout of houses and roads. This minimizes the cost of the roads to be laid.

Efficient Sewage Pipeline Layouts

This problem is similar to the above problem. The outlets can be treatment facilities whereas the logical nodes or operations are the houses which generate sewage or junctions of sewage pipes. The above program can similarly be used to find the layout of sewage pipes which minimizes the cost.

Code Links:

BellMan Ford
EdmondsKarp
ExpressionTree
CycleCancellation
LPSolving
Linker(Main)

Images

Sample Output
Expression Tree
Bipartite Graph