Bellman Ford Algorithm

This is an algorithm that solves the single source shortest path problem (sssp) (finds the distances and shortest paths from a source to all other nodes) for the case where the weights can be negative.

If there is a negative cycle, the shortest path and the distance are not well defined. The algorithm returns "a negative cycle exists".

We will see an application for Solving a *System of Difference Constraints*

Bellman-Ford's Algorithm

- The idea:
  - There is a shortest path from *s* to any other vertex that does not contain a non-negative cycle (can be eliminated to produce a lighter path).
  - The maximal number of edges in such a path with no cycles is |*V*|-1, because it can have at most |V| nodes on the path if there is no cycle.
  - $\Rightarrow$ it is enough to check paths of up to |*V*|-1 edges

Bellman-Ford's Algorithm

Initialize(G, s)

for $i \leftarrow 1$ to $|V| - 1$

  for each edge $(v, w) \in E[G]$

    do if dist(w) > dist(v) + $c(v, w)$

      $dist(w) = dist(v) + c(v, w)$

      $Parent(w) = v$

for each edge $(v, w) \in E[G]$

  if $dist[w] > d[v] + c(v, w)$ return "negative cycle"

Bellman-Ford's Algorithm

- The first pass over the edges – only neighbors of *s* are affected (1 edge paths). All shortest paths with one edge are found.

- The second pass – shortest paths of 2 edges are found

- After |*V*|-1 passes, all possible paths are checked.
- We claim: we need to update any vertex in the last pass if and only if there is a negative cycle reachable from *s* in *G.*

Bellman Ford Algorithm

- One direction we already know: if we need to update an edge in the last iteration then there is a negative cycle, because we proved before that if there are no negative cycles, and the shortest paths are well defined, we find them in |V|-1 iteration.
- We claim that if there is a negative cycle, we will discover a problem in the last iteration. Because, suppose there is a negative cycle

$$v_1, \ldots . v_{k-1}, v_k = v_1$$

But the algorithm does not find any problem in the last iteration, which means that for all edges, we have that $\mathrm{dist}(\mathrm{w}) \leq \mathrm{dist}(\mathrm{v}) + c(v, w)$

So the inequality is true for all edges in the cycle. So…

- We can write for all edges in the cycle

$$\mathrm{dist}(\mathrm{v}_2) \le \mathrm{dist}(\mathrm{v}_1) + c(v_2, v_1)$$

$$\mathrm{dist}(\mathrm{v}_3) \le \mathrm{dist}(\mathrm{v}_2) + c(v_3, v_2)$$

$$\ldots$$

$$\mathrm{dist}(\mathrm{v}_k) \le \mathrm{dist}(\mathrm{v}_{k\text{-}1}) + c(v_k, v_{k-1})$$

$$\sum_{i=2}^{k} dist(v_i) \le \sum_{1}^{k-1} dist(v_i) + \sum_{1}^{k-1} c(v_k, v_{k-1})$$

- After summing up over all edges in the cycle, we discover that the term on the left is equal to the first term on the right(just a different order of summation.) We can subtruct them, and we get that the cycle is actually positive, which is a contradiction.

Bellman-Ford Run Time

- The algorithm run time is:
  - Goes over v-1 vertexes, O(V)
  - For each vertex relaxation over E, O(E)
  - Altogether O(VE)

Application of Bellman-Ford

- The Bellman-Ford algorithm can be used to solve a set of difference constraints for n variables, like the following set of linear inequalities :

$$
\begin{aligned}
x_1 - x_2 &\le 0 \\
x_1 - x_5 &\le -1 \\
x_2 - x_5 &\le 1 \\
x_3 - x_1 &\le 5 \\
x_4 - x_1 &\le 4 \\
x_4 - x_3 &\le -1 \\
x_5 - x_3 &\le -3 \\
x_5 - x_4 &\le -3
\end{aligned}
$$

Application of Bellman-Ford

- There are many uses for a set of difference constraints, for instance:
  - The variables can represent the times of different events
  - The inequalities are the constraints over there synchronization.

Application of Bellman-Ford

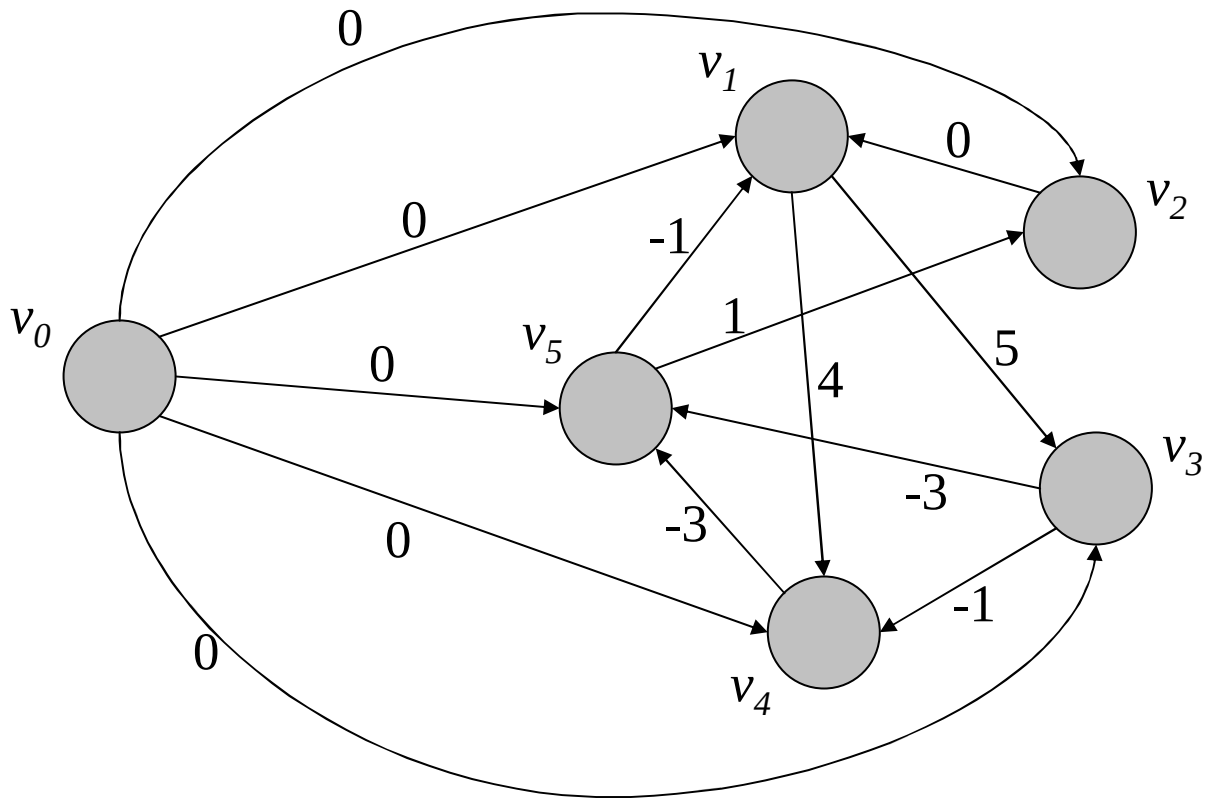
- What is the connection between the Bellman-Ford algorithm and a set of linear inequalities?
- We can interpret the problem as a directed graph.
- The graph is called the *constraint graph* of the problem
- After constructing the graph, we could use the Bellman-Ford algorithm.
- The result of the Bellman-Ford algorithm is the vector x that solves the set of inequalities.

Application of Bellman-Ford

- Building the *constraint graph:*
  - Each variable x_i corresponds to a node v_i
  - Each constraint $x_j - x_i \leq b_{ij}$ corresponds to an edge from v_i to v_j with weight b_{ij}
  - We add a special node v_o and we add edges from this special node to all other nodes. The weights of these edges are 0.

Application of Bellman-Ford

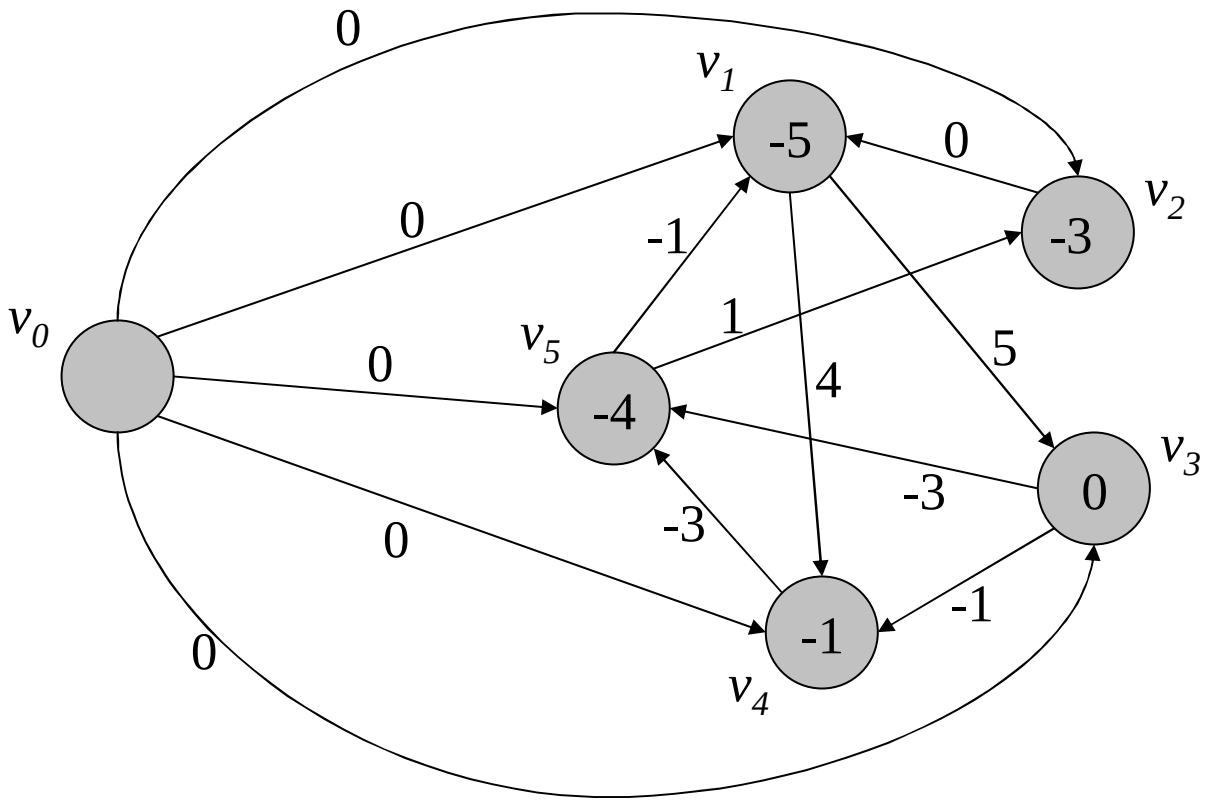
- The constraint graph:



- Note the similarity between the constraints on the variables, and the conditions for the path to be shortest in the Bellman Ford algorithm.
- We now apply the Bellman Ford algorithm on the graph, with the source being the extra node v_o

Application of Bellman-Ford

- The solution:



Application of Bellman-Ford

Claim: We claim that if Bellman Ford did not find a negative cycle, and we substitute $x_i = dist(v_i)$

where dist is the output of the Bellman-Ford algorithm, then the x's satisfy the set of constraints.

We also claim that if Bellman Ford found a negative cycle, then there is no solution for the set of constraints!

We will prove these two claims in the next slide.

- Suppose Bellman Ford did not find a cycle, and output a solution dist(v1),dist(v2)….
- So at the final iteration of the algorithm, all edges satisfied

$$\mathrm{dist}(\mathrm{w}) \le \mathrm{dist}(\mathrm{v}) + c(v,w)$$

- So our choice $x_i = dist(v_i)$ means that all constraints are satisfied.
- Now if Bellman-Ford did find a cycle, then we claim there can not be a solution. Because if there is a solution, $x_1,\ldots x_n$ which satisfies the set of constraints, we have $x_j - x_i \le b_{ij}$ for all the constraints. Among all these inequalities, let us just pick those which correspond to the edges in the negative cycle in the graph. Summing all these constraints we find just like before that the cycle is non negative, so we get a contradiction as before.

Application of Bellman Ford

- This shows that the Bellman Ford algorithm finds a solution for the set of difference constraints, and if there is no solution, it tells us about it.
- Note that if we found a solution, adding to all variables a constant c still gives a solution.
- This is an application of the advantage of the graph data structure in a slightly less expected place than we saw before.