Dynamic Programming | Set 23 (Bellman–Ford Algorithm)

Given a graph and a source vertex *src* in graph, find shortest paths from *src* to all vertices in the given graph. The graph may contain negative weight edges.
We have discussed Dijkstra's algorithm for this problem. Dijksra's algorithm is a Greedy algorithm and time complexity is O(VLogV) (with the use of Fibonacci heap). *Dijkstra doesn't work for Graphs with negative weight edges, Bellman-Ford works for such graphs. Bellman-Ford is also simpler than Dijkstra and suites well for distributed systems. But time complexity of Bellman-Ford is O(VE), which is more than Dijkstra.*

Algorithm
Following are the detailed steps.

Input: Graph and a source vertex *src*

Output: Shortest distance to all vertices from *src*. If there is a negative weight cycle, then shortest distances are not calculated, negative weight cycle is reported.

1) This step initializes distances from source to all vertices as infinite and distance to source itself as 0. Create an array dist[] of size |V| with all values as infinite except dist[src] where src is source vertex.

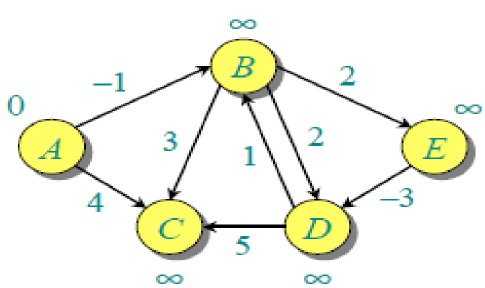
2) This step calculates shortest distances. Do following |V|-1 times where |V| is the number of vertices in given graph.
…..**a)** Do following for each edge u-v
………………If dist[v] > dist[u] + weight of edge uv, then update dist[v]
………………….dist[v] = dist[u] + weight of edge uv

3) This step reports if there is a negative weight cycle in graph. Do following for each edge u-v
……If dist[v] > dist[u] + weight of edge uv, then "Graph contains negative weight cycle"
The idea of step 3 is, step 2 guarantees shortest distances if graph doesn't contain negative weight cycle. If we iterate through all edges one more time and get a shorter path for any vertex, then there is a negative weight cycle

How does this work? Like other Dynamic Programming Problems, the algorithm calculate shortest paths in bottom-up manner. It first calculates the shortest distances for the shortest paths which have at-most one edge in the path. Then, it calculates shortest paths with at-nost 2 edges, and so on. After the ith iteration of outer loop, the shortest paths with at most i edges are calculated. There can be maximum |V| – 1 edges in any simple path, that is why the outer loop runs |v| – 1 times. The idea is, assuming that there is no negative weight cycle, if we have calculated shortest paths with at most i edges, then an iteration over all edges guarantees to give shortest path with at-most (i+1) edges (Proof is simple, you can refer this or MIT Video Lecture)

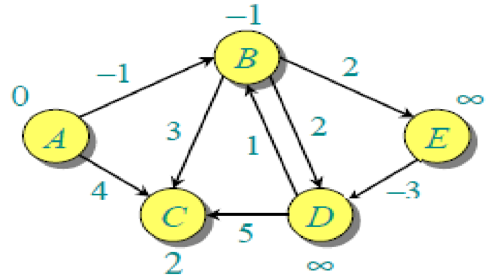
Example
Let us understand the algorithm with following example graph. The images are taken from this source.

Let the given source vertex be 0. Initialize all distances as infinite, except the distance to source itself. Total number of vertices in the graph is 5, so *all edges must be processed 4 times.*



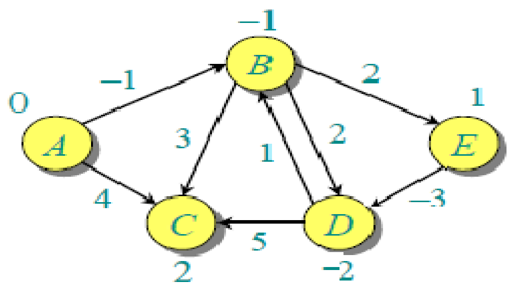
A	B	C	D	E
0	∞	∞	∞	∞

Let all edges are processed in following order: (B,E), (D,B), (B,D), (A,B), (A,C), (D,C), (B,C), (E,D). We get following distances when all edges are processed first time. The first row in shows initial distances. The second row shows distances when edges (B,E), (D,B), (B,D) and (A,B) are processed. The third row shows distances when (A,C) is processed. The fourth row shows when (D,C), (B,C) and (E,D) are processed.



A	B	C	D	E
0	∞	∞	∞	∞
0	−1	∞	∞	∞
0	−1	4	∞	∞
0	−1	2	∞	∞

The first iteration guarantees to give all shortest paths which are at most 1 edge long. We get following distances when all edges are processed second time (The last row shows final values).



A	B	C	D	E
0	∞	∞	∞	∞
0	−1	∞	∞	∞
0	−1	4	∞	∞
0	−1	2	∞	∞
0	−1	2	∞	1
0	−1	2	1	1
0	−1	2	−2	1

The second iteration guarantees to give all shortest paths which are at most 2 edges long. The algorithm processes all edges 2 more times. The distances are minimized after the second iteration, so third and fourth iterations don't update the distances.

Implementation:

```
// A C / C++ program for Bellman-Ford's single source shortest path algorithm

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <limits.h>

// a structure to represent a weighted edge in graph
struct Edge
{
    int src, dest, weight;
};

// a structure to represent a connected, directed and weighted graph
struct Graph
{
    // V-> Number of vertices, E-> Number of edges
    int V, E;

    // graph is represented as an array of edges.
    struct Edge* edge;
};

// Creates a graph with V vertices and E edges
```

```
struct Graph* createGraph(int V, int E)
{
    struct Graph* graph = (struct Graph*) malloc( sizeof(struct Graph) );
    graph->V = V;
    graph->E = E;

    graph->edge = (struct Edge*) malloc( graph->E * sizeof( struct Edge ) );

    return graph;
}

// A utility function used to print the solution
void printArr(int dist[], int n)
{
    printf("Vertex   Distance from Source\n");
    for (int i = 0; i < n; ++i)
        printf("%d \t\t %d\n", i, dist[i]);
}

// The main function that finds shortest distances from src to all other
// vertices using Bellman-Ford algorithm.  The function also detects negative
// weight cycle
void BellmanFord(struct Graph* graph, int src)
{
    int V = graph->V;
    int E = graph->E;
    int dist[V];

    // Step 1: Initialize distances from src to all other vertices as INFINIT
    for (int i = 0; i < V; i++)
        dist[i]   = INT_MAX;
    dist[src] = 0;

    // Step 2: Relax all edges |V| - 1 times. A simple shortest path from src
    // to any other vertex can have at-most |V| - 1 edges
    for (int i = 1; i <= V-1; i++)
    {
        for (int j = 0; j < E; j++)
        {
            int u = graph->edge[j].src;
            int v = graph->edge[j].dest;
            int weight = graph->edge[j].weight;
            if (dist[u] != INT_MAX && dist[u] + weight < dist[v])
                dist[v] = dist[u] + weight;
        }
    }

    // Step 3: check for negative-weight cycles.  The above step guarantees
    // shortest distances if graph doesn't contain negative weight cycle.
    // If we get a shorter path, then there is a cycle.
    for (int i = 0; i < E; i++)
    {
        int u = graph->edge[i].src;
```

```
        int v = graph->edge[i].dest;
        int weight = graph->edge[i].weight;
        if (dist[u] != INT_MAX && dist[u] + weight < dist[v])
            printf("Graph contains negative weight cycle");
    }

    printArr(dist, V);

    return;
}

// Driver program to test above functions
int main()
{
    /* Let us create the graph given in above example */
    int V = 5;  // Number of vertices in graph
    int E = 8;  // Number of edges in graph
    struct Graph* graph = createGraph(V, E);

    // add edge 0-1 (or A-B in above figure)
    graph->edge[0].src = 0;
    graph->edge[0].dest = 1;
    graph->edge[0].weight = -1;

    // add edge 0-2 (or A-C in above figure)
    graph->edge[1].src = 0;
    graph->edge[1].dest = 2;
    graph->edge[1].weight = 4;

    // add edge 1-2 (or B-C in above figure)
    graph->edge[2].src = 1;
    graph->edge[2].dest = 2;
    graph->edge[2].weight = 3;

    // add edge 1-3 (or B-D in above figure)
    graph->edge[3].src = 1;
    graph->edge[3].dest = 3;
    graph->edge[3].weight = 2;

    // add edge 1-4 (or A-E in above figure)
    graph->edge[4].src = 1;
    graph->edge[4].dest = 4;
    graph->edge[4].weight = 2;

    // add edge 3-2 (or D-C in above figure)
    graph->edge[5].src = 3;
    graph->edge[5].dest = 2;
    graph->edge[5].weight = 5;

    // add edge 3-1 (or D-B in above figure)
    graph->edge[6].src = 3;
    graph->edge[6].dest = 1;
    graph->edge[6].weight = 1;
```

```
    // add edge 4-3 (or E-D in above figure)
    graph->edge[7].src = 4;
    graph->edge[7].dest = 3;
    graph->edge[7].weight = -3;

    BellmanFord(graph, 0);

    return 0;
}
```

Output:

```
Vertex   Distance from Source
0                0
1                -1
2                2
3                -2
4                1
```

Notes
1) Negative weights are found in various applications of graphs. For example, instead of paying cost for a path, we may get some advantage if we follow the path.

2) Bellman-Ford works better (better than Dijksra's) for distributed systems. Unlike Dijksra's where we need to find minimum value of all vertices, in Bellman-Ford, edges are considered one by one.

Exercise
1) The standard Bellman-Ford algorithm reports shortest path only if there is no negative weight cycles. Modify it so that it reports minimum distances even if there is a negative weight cycle.

2) Can we use Dijksra's algorithm for shortest paths for graphs with negative weights – one idea can be, calculate the minimum weight value, add a positive value (equal to absolute value of minimum weight value) to all weights and run the Dijksra's algorithm for the modified graph. Will this algorithm work?

References:
http://www.youtube.com/watch?v=Ttezuzs39nk
http://en.wikipedia.org/wiki/Bellman%E2%80%93Ford_algorithm
http://www.cs.arizona.edu/classes/cs445/spring07/ShortestPath2.prn.pdf

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

Related Topics:

- Assign directions to edges so that the directed graph remains acyclic
- K Centers Problem | Set 1 (Greedy Approximate Algorithm)
- Find the minimum cost to reach destination using a train
- Applications of Breadth First Traversal
- Optimal read list for given number of days

- Print all paths from a given source to a destination
- Minimize Cash Flow among a given set of friends who have borrowed money from each other
- Boggle (Find all possible words in a board of characters)

Tags: Dynamic Programming, Graph

Writing code in comment? Please use **ideone.com** and share the link here.

sourcedelica • 2 days ago

What's the algorithm for reporting minimum distances even if there is a negative weight cycle?

• Reply • Share ›

Ach Ref • 25 days ago

#GeekforGeeks please update this Post this solution is an improvement of the original DP solution is not a DP solution :)

• Reply • Share ›

Guest • a month ago

this is not Dynamic programming

watch this MIN 17

https://www.youtube.com/watch?...

• Reply • Share ›

Ach Ref → Guest • 24 days ago

yes this is an improvement of original DP solution of bellman ford

• Reply • Share ›

codelearner • 2 months ago

for the step 3 portion of this, it seems to figure out if it is a negative edge but not if it is a cycle or not!!! or am i missing something !!

• Reply • Share ›

Krishna • 2 months ago

I think this link gives a better explanation and simpler code.... https://theoryofprogramming.wo... ... It helps..!

1 • Reply • Share ›

Guest • 2 months ago

is that DP o.O i think this code

Create a table DP of size n × n.
● Set DP[v][0] = ∞ for all v ≠ s.
● Set DP[s][0] = 0
● For i = 1 to n – 1, for all v ∈ V:
– Set DP[v][i] =
min {
DP[v][i – 1],
min { DP[u][i – 1] + w(u, v) } (where (u, v) ∈ E)
}
● Return row n of DP

• Reply • Share ›

Shiva Amrit • 3 months ago

Can someone please clarify my doubt ?

In the void BellmanFord(struct Graph* graph, int src) function , inside the for loop , in the if condition, why are we checking " dist[u] != INT_max" . The code is working without this check also .

• Reply • Share ›

sd • 7 months ago

Dijkstra works for negative weight edges, not for negative weight cycle.

1 • Reply • Share ›

Aditya Goel → sd • 3 months ago

No, it doesn't work for either of the two.

• Reply • Share ›

Nitin Maheshwari • 7 months ago

Hi I found a case for which this algo is not working :

verticesCount =3 edgesCount=4
src dest weight
0 1 -1

1 2 2

0 2 3

1 0 1

Please update this algo

• Reply • Share ›

arjomanD • a year ago

wait,why just update v's distance ? why don't check u's distance too ? (for both endpoints of edge i mean)

• Reply • Share ›

GOPI GOPINATH arjomanD • a year ago

we already reached 'u', It means we already computed distance of 'u'

• Reply • Share ›

prashant jha • a year ago

u can also implement it with queue

http://ideone.com/3BRwwB

for cycle with negative weight sum it will run into infinte loop because each tym a new path is possible...so if a vertex is inserted more than n times in the queue u can say graph has a negative weight cycle

1 • Reply • Share ›

Sidharth prashant jha • a year ago

Do u know the compexity of this solution ?

• Reply • Share ›

Lohith Ravi • a year ago

Can some one tell me why Dijikstras do not work for negatives. I just applied dijikstras on this and got the same output.

• Reply • Share ›

Siddharth Thevaril Lohith Ravi • 3 months ago

Dijstra's algorithm **MAY** work for some graphs with negative edges.

• Reply • Share ›

GOPI GOPINATH Lohith Ravi • a year ago

Try to apply Dijkstras on the graph and see the result.

http://i.stack.imgur.com/xp1H4...

• Reply • Share ›

tokes • a year ago

There is a problem where if you change the source node to something different from "a" which is "0" you will get wrong answers.

• Reply • Share ›

viki • 2 years ago

Hi Sir,

Replace the code "dist[u] + weight < dist[v]"
by "dist[u] < dist[v] - weight" to overcome integer overflow.

BTW excellent post.

1 • Reply • Share ›

amrit viki • 3 months ago

can you please elaborate as to why an overflow error will occur ??

1 • Reply • Share ›

Ravi Guru viki • 7 months ago

really good

• Reply • Share ›

Hero viki • a year ago

Yes， that might be a overflow problem, but still your way doesn't overcome overflow problem. We should do like this:

//handle overflow

if (dist[i] != INT_MAX && dist[i] + w < dist[j])

{

dist[j] = dist[i] + w;

}

• Reply • Share ›

GeeksforGeeks Mod Hero • 7 months ago

@All Thanks for pointing out the overflow problem. We have updated the cod to handle the same.

• Reply • Share ›

Ach Ref GeeksforGeeks • 24 days ago

#GeekforGeeks Hi the solution is correct but this not A DP solution this an improvement of the original DP solution of bellman ford algorithm

• Reply • Share ›

Krishna Prasad • 2 years ago

For step 2:
what if i use....

int count =0 flag=1;

```
int count =0,flag=1;

while(flag)
flag=0;
for (int j = 0; j < E; j++)
{
int u = graph->edge[j].src;
int v = graph->edge[j].dest;
int weight = graph->edge[j].weight;
if (dist[u] + weight < dist[v])
{ dist[v] = dist[u] + weight;
flag=1;}

}
count++;
if(count>V)
```

see more

1 • Reply • Share ›

Kura Desta • 2 years ago

hi, please support me, how to solve problems using forward and backward recursive methods. thanks

• Reply • Share ›

???? ??????? • 2 years ago

ok thanks alot for all efforts.

• Reply • Share ›

coder • 2 years ago

the corresponding mit video lecture is great

• Reply • Share ›

Kumar Vikram • 2 years ago

Another implementation of the above problem using adjacency list..

```
[sourcecode language="C++"]
#include<iostream>
#include <list>
#include <stack>
#include <limits.h>

using namespace std;

class AdjListNode
{
int v;
```

int weight;
public:
AdjListNode(int _v, int _w) { v = _v; weight = _w;}
int getV() { return v; }
int getWeight() { return weight; }
};

see more

• Reply • Share ›

sumit → Kumar Vikram • 2 years ago

@Kumar Vikram , there is one major issue with your approach above , the algo talks about iterating through each edge V-1 times . what you have done in bell_ford() is to iterate through all edges as per the adj list just once . may be you need to change the logic a bit to accommodate this ...

• Reply • Share ›

raghson • 2 years ago

In the if condition
if(dist[u] + weight < dist[v]) :

the dist[v] is updated even in the case when dist[u] is INT_MAX, dist[v] is INT_MAX and weight is negative as the condition is satisfied. So, the if-condition should be modified to if(dist[u] + weight < dist[v] && (dist[u]!=INT_MAX)). It will save time which is spent calculating dist[v] in the cases which are of type mentioned above. Please correct me if I am wrong.

2 • Reply • Share ›

Kumar Vikram • 2 years ago

this is an implementation of the given program with the addition that it also calculates the minimum path along with the minimum path length.

```
 /*// A C / C++ program for Bellman-Ford's single source shortest path algorithm.

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <limits.h>



struct Edge
{
    int src, dest, weight;
};
```

```
struct Graph
```

see more

2 • Reply • Share ›

Kumar Vikram • 2 years ago

```
 this is an implementation of above program that also calculates the shortest path along wi
/* Paste your code here (You may delete these lines if not writing code) */
```

// A C / C++ program for Bellman-Ford's single source shortest path algorithm.

#include
#include
#include
#include

struct Edge
{
int src, dest, weight;
};

struct Graph
{

see more

• Reply • Share ›

Shashank • 2 years ago

The algorithm is correct but there is a small glitch in the explanation.

The idea is, if we have calculated shortest paths with at most i edges, then an iteration over all edges guarantees to give shortest path with at-most (i+1) edges

My Point:
After 'i' iterations every node does not hold a value for shortest path with at most i edges.

It holds a value which is 'at-most' the value for the shortest path with at most i edges. But it definitely holds a value for some path. It is just that after 'i' iterations it may hold a value for a path with greater than 'i' edges.

This can also be seen in the example cited above where -
Node C holds the matrix value 2 after 1st iteration when the value 2 for Node C is for a path with 2 edges. This happens because of the sequence in which the nodes are considered.

Just a minor correction in the description :)

```
/* Paste your code here (You may delete these lines if not writing code) */
```

^ | v • Reply • Share ›

Ach Ref → Shashank • 2 months ago

yes :/ this is the solution

Create a table DP of size n × n.
● Set DP[v][0] = ∞ for all v ≠ s.
● Set DP[s][0] = 0
● For i = 1 to n – 1, for all v ∈ V:
– Set DP[v][i] =
min {
DP[v][i – 1],
min { DP[u][i – 1] + w(u, v) } (where (u, v) ∈ E)
}
● Return row n of DP

^ | v • Reply • Share ›

- Interview Experiences
- Advanced Data Structures

  - Dynamic Programming
  - Greedy Algorithms
  - Backtracking
  - Pattern Searching
  - Divide & Conquer
  - Mathematical Algorithms
  - Recursion
  - Geometric Algorithms
-

- ## Popular Posts

  - All permutations of a given string
  - Memory Layout of C Programs
  - Understanding "extern" keyword in C
  - Median of two sorted arrays
  - Tree traversal without recursion and without stack!
  - Structure Member Alignment, Padding and Data Packing
  - Intersection point of two Linked Lists
  - Lowest Common Ancestor in a BST.
  - Check if a binary tree is BST or not
  - Sorted Linked List to Balanced BST
- Follow @GeeksforGeeks

- ## Recent Comments

  - lt_k

    i need help for coding this function in java...

    Java Programming Language · 1 hour ago
  - Piyush

    What is the purpose of else if (recStack[*i])...

    Detect Cycle in a Directed Graph · 1 hour ago
  - Andy Toh

    My compile-time solution, which agrees with the...

    Dynamic Programming | Set 16 (Floyd Warshall Algorithm) · 1 hour ago
  - lucy

    because we first fill zero in first col and...

    Dynamic Programming | Set 29 (Longest Common Substring) · 2 hours ago

- lucy

  @GeeksforGeeks i don't n know what is this long...

  Dynamic Programming | Set 28 (Minimum insertions to form a palindrome) · 3 hours ago

- manish

  Because TAN is not a subsequence of RANT. ANT...

  Given two strings, find if first string is a subsequence of second · 3 hours ago