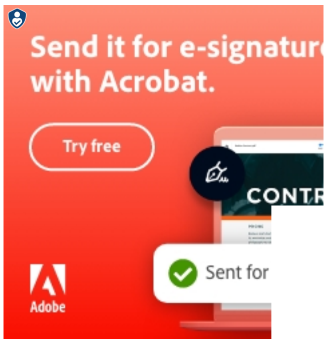
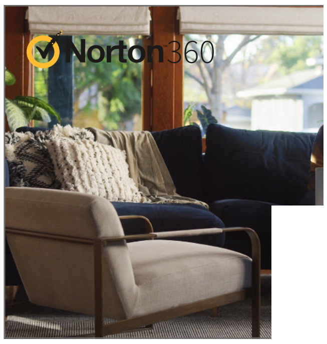
Question: In each of the following, a greedy algorithm is proposed for t...

(3 bookmarks)

In each of the following, a greedy algorithm is proposed for the given problem. Give a simple counterexample for each of them to show that the proposed greedy algorithm does not always return the optimal solution. You should give an example input, state the solution returned by the greedy algorithm on that input, and another solution that is better than the greedy solution for that input.
(a) Input: An undirected complete graph G (i.e. there is an edge (with weight) between any two vertices in the graph); a starting vertex s and a destination vertex t in G.
Problem: Find a path that begins at s, ends at t, and visits every other vertex in G exactly once, and such that the total weight of this path is as small as possible.
Algorithm: Let u be the current vertex (which initially is s). Find the minimum-weight edge among all edges (u; v) where v is neither t nor any vertex already visited. Add this edge to the path, and update the current vertex to v. Repeat this until t is the only vertex not yet visited. At this point add the edge from the current vertex to t as the nal edge of the path. [10 marks]
(b) Input: A set Y of Utube channels, a set P of people and a and a set of pairs (yi; pj) indicating person pj subscribed channel yi.
Problem: Choose a subset Y 0 of Utube channels from Y so that everyone in P sub-scribes to at least one channel in Y 0, and that the number of channels in Y 0 is as small as possible.
Algorithm: Repeatedly add to Y 0 the channel that would give the largest number of `new' subscribers (i.e. those who did not subscribe to any channel already in Y 0), until everyone in P has subscribed something in Y 0. [10 marks]
(c) Input: A set of objects, each with a weight between 0 and 1.
Problem: Put all objects into a number of containers, where each container can hold objects of total weight at most 1, and such that the number of containers used is minimised.
Algorithm: Sort all objects in decreasing order of weights. For each object in this sorted order, among all existing containers that would t this object, put it into the one with the largest total weight of objects already in there. If no existing container can t this object in, put it into a new container.





Expert Answer

Vishal Singh answered this
454 answers

Was this answer helpful? 1 0

EXPLANATION:

How greedy algorithms work

Greedy algorithms always choose the best available option.

In general, they are computationally cheaper than other families of algorithms like dynamic programming, or brute force. This is because they don't explore the solution space too much. And, for the same reason, they don't find the best solution to a lot of problems.

Answer a)

This algorithm finds such a path by always going to the nearest vertex. That's why we say it is a greedy algorithm.

This is pseudocode for the algorithm. I denote with `G` the graph and with `s` the source node.

Forum Donate

When to Use Greedy Algorithms – And When to Avoid Them [With Example Problems]

Jose J. Rodríguez

Greedy algorithms try to find the optimal solution by taking the best available choice at every step.

For example, you can greedily approach your life. You can always take the path that maximizes your happiness today. But that doesn't mean you'll be happier tomorrow.

Similarly, there are problems for which greedy algorithms don't yield the best solution. Actually, they might yield the worst possible solution.

But there are other cases in which we can obtain a solution that is good enough by using a greedy strategy.

In this article, I'll write about greedy algorithms and the use of this strategy even when it doesn't guarantee that you'll find an optimal solution.

The first section is an introduction to greedy algorithms and well-known problems that are solvable using this strategy. Then I'll talk about problems in which the greedy strategy is a really bad option. And finally, I'll show you an example of a good approximation through a greedy algorithm.

Note: Most of the algorithms and problems I discuss in this article include graphs. It would be good if you are familiar with graphs to get the most out of this post.

How greedy algorithms work

Greedy algorithms always choose the best available option.

In general, they are computationally cheaper than other families of algorithms like dynamic programming, or brute force. This is because they don't explore the solution space too much. And, for the same reason, they don't find the best solution to a lot of problems.

But there are lots of problems that are solvable with a greedy strategy, and in those cases that strategy is precisely the best way to go.

One of the most popular greedy algorithms is Dijkstra's algorithm that finds the path with the minimum cost from one vertex to the others in a graph.

This algorithm finds such a path by always going to the nearest vertex. That's why we say it is a greedy algorithm.

This is pseudocode for the algorithm. I denote with G the graph and with s the source node.

```
Algorithm(G, s):
  distances <- list of length equal to the number of nodes of the graph, initially it has all its elements equal to infinite

  distances[s] = 0

  queue = the set of vertices of G

  while queue is not empty:

      u <- vertex in queue with min distances[u]
```

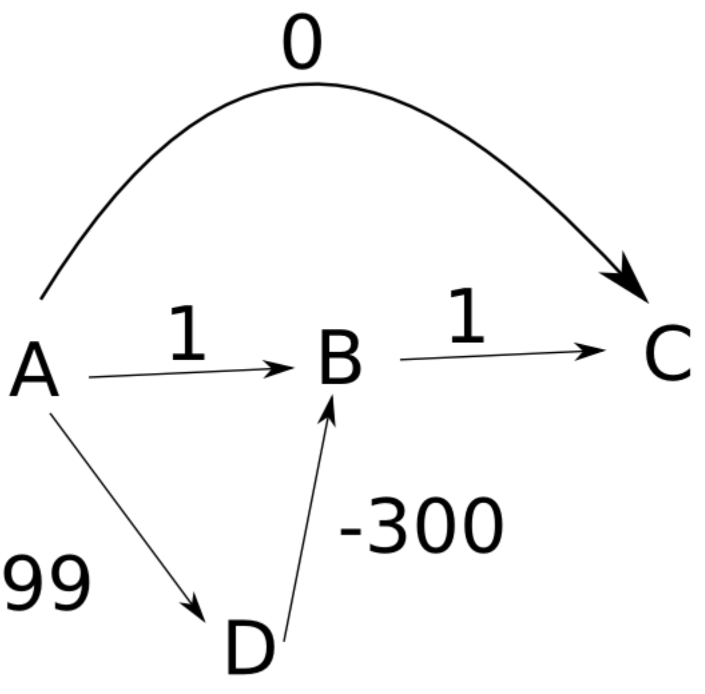
```
        temp = distances[u] + value(u,v)

        if temp < distances[v]:
            distances[v] = temp
    return distances
```

After running this algorithm, we get a list of `distances` such that `distances[u]` is the minimum cost to go from node `s` to node `u`.

This algorithm is guaranteed to work only if the graph doesn't have edges with negative costs. A negative cost in an edge can make the greedy strategy choose a path that is not optimal.

And the problem is even bigger. I can always build a graph with negative edges in a way that Dijkstra's solution would be as bad as I wanted! Consider the following example that was extracted from Stackoverflow



$$\begin{tikzcd}
A \arrow[r, "1"] \arrow[rr, bend left, "0"] \arrow[dr, "99"'] & B \arrow[r, "1"] & C \\
& D \arrow[u, "-300"'] &
\end{tikzcd}$$

algorithm fails to find the distance between `A` and `C`. It finds `d(A, C) = 0` when it should be -200. And if we decrease the value of the edge `D -> B`, we'll obtain a distance that we'll be even farther from the actual minimum distance.

It is recommended to post multiple questions seperately as I can only answer 1 question at a time.

View comments (1)

Practice with similar questions

Q: In each of the following, a greedy algorithm is proposed for the given problem. Give a simple counterexample for each of them to show that the proposed greedy algorithm does not always return the optimal solution. You should give an example input, state the solution returned by the greedy algorithm on that input, and another solution that is better than the greedy solution for that...

A: See answer

the proposed greedy algorithm does not always return the optimal solution (this is algorithm queations)

A: See answer 100% (1 rating)

Q: In each of the following, a greedy algorithm is proposed for the given problem. Give a simple counterexample for each of them to show that the proposed greedy algorithm does not always return the optimal solution. You should give an example input, state the solution returned by the greedy algorithm on that input, and another solution that is better than the greedy solution for that...

A: See answer

Show more