Immagine che contiene design

Descrizione generata automaticamente con attendibilità bassa

Advanced Algorithms: Exercises Swiss Knife

Immagine che contiene simbolo, clipart, creatività

Descrizione generata automaticamente

Table of Contents

[2 Graphs 3](#_Toc164112024)

[2.1 Properties of graphs 3](#_Toc164112025)

[2.2 DFS Exercises 3](#_Toc164112026)

[2.3 Uniqueness of MSTs 4](#_Toc164112027)

[2.4 Kruskal Union-Find 6](#_Toc164112028)

[2.5 Dijkstra with Heaps 6](#_Toc164112029)

[2.6 Eulerian Circuit in Linear Time 7](#_Toc164112030)

[3 NP-Hardness 9](#_Toc164112031)

[3.1 Clique is NP-Hard 9](#_Toc164112032)

[3.2 Vertex Cover is NP-Hard 10](#_Toc164112033)

[3.3 More Reductions can be made 11](#_Toc164112034)

[4 Approximation Algorithms 13](#_Toc164112035)

[4.1 Lower Bound for Vertex Cover Greedy Algorithm 13](#_Toc164112036)

[4.2 Approximation Factor of Approx Vertex Cover 15](#_Toc164112037)

# Graphs

## Properties of graphs

*Let be a simple, connected graph with vertices and edges. Then:*

1. *is a tree*
2. *is connected*
3. *is acyclic (i.e., is a forest)*

*Prove the previous properties*.

Solution

1. In the summation, every edge is counted exactly twice
2. In a simple graph, there are possible pairs of vertices
3. Fix a root on a vertex (so, consider as rooted tree, thanks to the equivalence between rooted tree and “free” tree). Then represent father-child relationships, which are (which means each non-root node has a unique father)
4. is a tree that may have cycles it can only have more edges than a tree
   1. Consider connectivity removes edges and keeps the graph connected without cycles, thanks to edges
5. is a tree that may not be connected it can only have less edges than a tree
   1. If it is a tree without cycles, it is a forest, and its maximum edges are

## DFS Exercises

1. *Given a graph and two vertices determine, if it exists, a path from to*
2. *Given a graph return a cycle (if any)*

Solution

* 1st exercise ( path)
  + add a field .
  + Modify s.t. when a is labeled
    - then
  + Run . Check if has been visited
    - NO: then return “No path”
    - YES: starting from , follow the “parent” label, so as to build a path from to
  + Complexity: where is the number of edges of connected component
* 2nd exercise (cycle) 🡪 we go back thanks to back edges because they “close” the cycles
  + add a field and add a field
  + is a then
  + is a then
    - then is an ancestor of in the DFS tree
  + Run DFS on each connected component
  + Check all the edges
    - as soon as an edge is found as
    - and
    - then return a cycle adding to all the edges found in the path from to
    - if no is found, then return “No Cycles” (it would be a tree)

Complexity for both algorithms: 🡪 invoked DFS once for each connected component

## Uniqueness of MSTs

Exercise (uniqueness of MSTs):

*Show that if the weights of the edges are all distinct then there exists exactly one MST.*

(*Hint: cut and paste argument – similar to the theorem correctness*)

Immagine che contiene design, bicicletta

Descrizione generata automaticamente con attendibilità bassaSolution (with details of lesson but also other including Wikipedia and other sources)

Assume there are two MST different from each other, so the contrary and so an edge in one but not in the other; since weights are distinct, with min weight, call it , without loss of generality (not introducing any artificial assumption), and the argument is a cut-and-paste one (this choice will be unique, considering edge weights are all distinct from each other):

* add to this creates a cycle ; is (M)ST no cycles has an edge

* + because was chosen as the unique lowest-weight edge (only edge with minimum weight not in the other) among those belonging to exactly one of and
  + therefore the weight of must be greater than the weight of
* remove from get a new spanning tree with weight (so, smaller weight): contradiction, because is an MST!

Two conclusions can be done:

* more generally, if the edge weights are not all distinct then only the (multi-)set of weights in minimum spanning trees is certain to be unique; it is the same for all minimum spanning trees
* when the edge weights are not all distinct, it's possible for multiple different MSTs to exist
  + however, while the actual arrangement of edges in these MSTs may vary, the set of weights of the edges across all MSTs will remain the same
* conversely, *if weights are not all distinct, generally multiple MSTs can exist*

Other exercises

1. *Is the converse true? (e.g., are weights necessarily unique for every possible graph and this has to hold for every graph)*

Solution

No: think of as a tree (literally only thing professor will write – lame, I know, I added more).

A connected graph with repeated edge weights and this can still have a unique minimum spanning tree. Considered the trivial example of being a tree; in this case, there are no cycles, so any spanning tree will be minimal, hence unique, regardless of repeated edge weights.

In conclusion, we might say:

* Distinct weights guarantee a unique MST
* Repeated weights can have multiple MSTs
  + but the set of weights used will always be the same across all of them

1. *Show that the second best MST, that is, the spanning tree of second-smallest total weight, is not necessarily unique (here we look for only one graph)*

Solution

Immagine che contiene calligrafia, diagramma, linea, Carattere

Descrizione generata automaticamenteThere will be a unique MST, but for the second best, according to where the cut will be displaced, there will definitely be more than one, given the cut can be done on more than two edges at a time.

If you want a complete formal explanation, see the book solution to this exercise [here](https://viterbi-web.usc.edu/~shanghua/teaching/Spring2010/public_html/files/HW2_Solutions_A.pdf) (look for problem B in the link).

## Kruskal Union-Find

*Argue that the complexity of (and of ) is .*

Solution

Initially, . can only increase because of a Union in which the root of the tree of points to another root (depth increases by 1 by construction). This happens only when the tree of gets merged to a tree of size not smaller (at least as big) when the depth of increases, the size of the tree of at least doubles.

* How many times can this happen?
  + times (at most)
    - therefore the depth of cannot increase more than times

So, we have two different algorithms with complexity . To reach complexity is still an open problem (there are slightly faster algorithms, but not not others optimal able to reach ).

## Dijkstra with Heaps

*Write an implementation of Dijkstra’s algorithm with heaps.*

Solution

(almost identical to Prim’s implementation with heaps)



(This algorithm gives only the length of the path, but it’s not difficult to also insert the actual path inside of this one)

*Complexity:*

* considering graph as adjacency list, vertices and edges
* iterations because of heap usage

Total number of operations: (there are operations on heaps)

## Eulerian Circuit in Linear Time

Problem:

*Given an undirected graph, an eulerian circuit is a cycle that traverses all the edges only once.*

*Show it can be solved in linear time.*

Solution

To solve the problem of finding an Eulerian circuit in an undirected graph in linear time, we can use the following algorithm:

1. Check if the graph is connected and has at most two vertices with odd degrees. If there are more than two vertices with odd degrees, then an Eulerian circuit cannot exist.
2. If there are exactly two vertices with odd degrees, start the Eulerian circuit at one of them. Otherwise, start from any vertex.
3. Traverse the graph using the following strategy:
   * At each vertex, choose an unvisited edge (if one exists) and traverse it.
   * If there are no unvisited edges at the current vertex, backtrack to the previous vertex.
4. If you can traverse all the edges and end up at the starting vertex, then an Eulerian circuit exists. Otherwise, an Eulerian circuit does not exist.

This algorithm works in linear time because it visits each edge exactly twice (once during the traversal and once during backtracking) and performs constant-time operations at each vertex. Therefore, the time complexity is , where is the number of vertices and is the number of edges in the graph.

There is an existing algorithm doing this done by *Hierholzer*, which showed the sufficient condition in Euler theorem, which states “a graph is connected if an only if has all nodes of even degree or if it has exactly two nodes of even degree”. It works for both directed and undirected graphs and works as follows:

*Preconditions*:

* All vertices in the graph must have even degrees

*Steps*:

* Start at any vertex (each one can be a starting point) and follow a trail of edges until returning to the starting vertex. This forms a partial circuit
* If the partial circuit covers all edges, the algorithm is complete. Otherwise, select any vertex in the current circuit that has unused edges and start a new circuit from that vertex, merging it into the previous circuit
* Repeat step 2 until all edges have been used
  1. At some point, we will visit a vertex and there will be no edges to follow
  2. Remember that Eulerian Cycle properties, every vertex should have even degrees or equal in-out degrees
  3. If we are stuck the first time it means that we formed a cycle, and the vertex that we are stuck on is the starting vertex. This means we returned where we started.
* The algorithm terminates when a complete Euler circuit is formed, where each edge is traversed exactly once, backtracking from the whole stack and holding complete knowledge of the structure

Hierholzer's algorithm has a linear runtime, making it an efficient method for finding an Euler circuit in a graph that meets the necessary requirements.

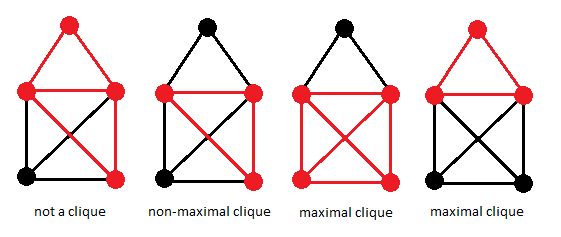
# NP-Hardness

Immagine che contiene testo, schermata, Carattere

Descrizione generata automaticamente

## Clique is NP-Hard

* (Maximum) Clique: compute the longest complete subgraph
  1. other name for a complete graph (from now on, the problem will be called Clique)
  2. below, a useful figure to clearly see the problem

*Show that Clique is NP-Hard*.

Solution (a nice graphical explanation [here](https://opendsa-server.cs.vt.edu/ODSA/Books/Everything/html/clique_to_independentSet.html))

Immagine che contiene schizzo, disegno, clipart

Descrizione generata automaticamenteDecision version:

* Input:
* Output: in a clique of size ?

We operate a reduction from Maximum Independent Set (Ham. circuit is not really related to it; as you can see [here](https://www.cs.cmu.edu/~avrim/451f11/lectures/lect1108.pdf), one can use 3SAT in order to show Clique is NP-complete). Figure here shows Independent Set.

* *Intuition*
  1. clique: vertices with all edges between them
  2. maximum independent set: vertices with no edges between them
* *Definition*
  1. given a graph , its edge-complement has the same vertex and an edge set such that (so, no common edges)
* *Observation*
  1. a set of vertices is independent in is a clique in the largest independent set in has the same size as the largest clique in

Immagine che contiene disegno, schizzo, diagramma, Line art

Descrizione generata automaticamenteTo make it super complete, let’s draw the schema of what we are doing – takes time, givemn the constant work needed to traverse all edges *and* vertices:

## Vertex Cover is NP-Hard

* (Minimum) Vertex Cover: compute the smallest vertex in a given graph
  1. From now on, only called Vertex Cover

*Show that Vertex Cover is NP-Hard*.

Solution (once again, a nice graphical explanation of this one [here](https://opendsa-server.cs.vt.edu/ODSA/Books/Everything/html/independentSet_to_vertexCover.html))

Decision version:

* Input:
* Output: in a vertex cover of size ?

We operate a reduction from Maximum Independent Set (once again, this is the most similar problem to the one we are proving)

* *Observation*
  1. a set of vertices is independent in is a vertex cover of
     1. in blue there is an independent set (actually the biggest one)
     2. Immagine che contiene linea, Arte bambini, diagramma, design

        Descrizione generata automaticamentethe other ones are the vertex cover

the longest independent set in has size , where is the size of the smallest vertex cover of

Independent set:

* Input:
* Output: in an independent set of size

Immagine che contiene calligrafia, schizzo, disegno, Line art

Descrizione generata automaticamenteOnce again, let’s represent this in a complete way:

## More Reductions can be made

Exercise

* *Show that*:

*these 3 problems are equivalent*.

Solution

1. Suppose that we have an efficient algorithm for solving Independent Set, it can simply be used to decide whether has a vertex cover of size at most , by asking it to determine whether G has an independent set of size at least

Given an instance of the Vertex Cover problem, consisting of a graph and an integer representing the size, we construct an instance of the Independent Set problem as follows:

1. Let (i.e., the graph for the Independent Set instance is the same as the original graph G).
2. Let (i.e., the target size of the independent set is the number of vertices in minus the size of the vertex cover ).

To show that this reduction is correct, we need to prove the following:

1. If has a vertex cover of size , then has an independent set of size .
2. If has an independent set of size , then G has a vertex cover of size .

Let’s prove both (1) and (2):

* Suppose is a vertex cover of size in . Then, the set is an independent set in (since covers all the edges, no two vertices in can be adjacent). Furthermore,
* Suppose is an independent set of size in . Then, the set is a vertex cover in (since is independent, every edge must have at least one endpoint in ). Furthermore, .

1. To show that , we need to provide a polynomial-time reduction from the Clique problem to the Vertex Cover problem. Here's one way to construct the reduction:

Given an instance of the Clique problem, consisting of a graph and an integer , we construct an instance of the Vertex Cover problem as follows:

1. Let (i.e., the graph for the Vertex Cover instance is the same as the original graph G).
2. Let (i.e., the target size of the vertex cover is the number of vertices in minus the size of the clique ).

To show that this reduction is correct, we need to prove the following:

1. If has a clique of size , then has a vertex cover of size .
2. If has a vertex cover of size , then has a clique of size .

Proof of (1): Suppose is a clique of size in . Then, the set is a vertex cover in (since is a clique, every edge must have at least one endpoint in ). Furthermore, .

Proof of (2): Suppose is a vertex cover of size in . Then, the set is a clique in (since is a vertex cover, every edge must have both endpoints in , which means is a clique). Furthermore, .

# Approximation Algorithms

## Lower Bound for Vertex Cover Greedy Algorithm

Very first algorithm you can think of? Use a *greedy approach*:

* select the vertex for the highest degree
* “remove” the touched edges
* repeat

Exercise: *show a LB on for this algorithm* – *the higher, the better* ( *is difficult*)

*(Hint: try to prove the best you can – it should be a constant factor)*

Solution

One possible idea is the following:

* take a round of vertices
* consider levels of vertices adding more

Immagine che contiene calligrafia, disegno, schizzo, diagramma

Descrizione generata automaticamenteWe call this problem: “degree-based greedy approximation for vertex cover”. Consider the following:

This image demonstrates a general idea for constructing a "bad" input instance to show a lower bound on the approximation ratio. The approach is to create a graph with multiple levels, where each level has more vertices than the previous level, but with fewer edges connecting to the next level.

The reasoning goes like this:

* Start with a single vertex (labeled "Greedy" in the image)
* At the next level, add a few vertices (e.g., 3) that are all connected to the first vertex
* At the next level, add more vertices (e.g., 8) that are only connected to the previous level vertices
* Continue adding more and more vertices at each level, with fewer connections to the previous level

The idea is that the greedy algorithm will pick all the vertices in the first level, then all the vertices in the second level, and so on, resulting in a large vertex cover. However, the optimal vertex cover would be to pick the intermediate level vertices, which can cover all the edges with fewer vertices.

The greedy algorithm, by design, will select the vertex with the highest degree at each step. This means:

* It will select the single vertex at the first level.
* Then, it will select all vertices at the second level (since they have the highest degree at that point).
* Next, it will select all vertices at the third level (since they are now the highest degree vertices remaining).

Therefore, the total size of the vertex cover produced by the greedy algorithm is:

However, the optimal vertex cover for this graph is to select the vertices at the second level. This covers all edges in the graph using only vertices.

By comparing the greedy solution size to the optimal solution size , we get an approximation ratio of:

(for large values of )

Immagine che contiene calligrafia, testo, diagramma, Arte bambini

Descrizione generata automaticamenteThe following considers a simpler idea instead:

1. In a bipartite graph where and are the two disjoint vertex sets, and contains edges only between and .
2. Consider the vertex in that has the maximum degree (i.e., connected to the most vertices in V).
3. The greedy algorithm will select and all its neighbors in .
4. However, the optimal solution is to select only the neighbors of in (and not itself).
5. This gives a lower bound on the approximation ratio .

The key observation is that by selecting the highest degree vertex in , the greedy algorithm is making the worst possible choice compared to the optimal solution of just selecting 's neighbors in This lower bound holds because:

* Greedy picks and vertices in , so size is
* Optimal just picks the vertices in that are neighbors of

So the approximation ratio is at least , which approaches as grows large.

## Approximation Factor of Approx Vertex Cover

The algorithm is:

*Complexity*:

Exercise: *show that the approximation factor of is exactly 2.*

Solution

Immagine che contiene calligrafia, manubrio, tipografia

Descrizione generata automaticamente

Consider the algorithm:

* The algorithm starts with an empty set (vertex cover set) and the original edge set . It iteratively selects an arbitrary edge ( from and adds both vertices and to the vertex cover set . It then removes all edges from that are incident on either or
* The algorithm continues this process until becomes empty, meaning all edges have been covered by the selected vertices in . Finally, it returns as the approximate vertex cover

The key observation is that for each edge selected, at least one of u or v must be present in the optimal vertex cover . This is because must cover all edges, and (u, v) is an edge in the original graph.

Therefore, during each iteration when an edge is processed, the algorithm adds at most two vertices to V', while the optimal vertex cover must contain at least one of these two vertices.

Consequently, we can establish the following inequality:

This inequality holds because the algorithm adds at most two vertices for each vertex that OPT must include to cover all edges.

Furthermore, we can construct an example where this bound is tight, meaning Consider a graph consisting of disconnected edges. In this case, the optimal vertex cover OPT would contain exactly one vertex from each edge, resulting in . However, the greedy algorithm would select both vertices for each edge, leading to

## Approx Vertex Cover edit to select only one vertex

aaa