Advanced Algorithms - Homework 3

Question 1

A

Assume we can solve this problem in polynomial time.

Let be a graph with the same vertices and edges but and, , the creation of is polynomial time.

If we can solve this problem in polynomial time hence, we can solve the **vertex cover** problem in poly time which is NP-complete it, contradiction! this problem is np-hard. Why it's working? for each vertex , , therefore we "blocking" the opportunity for any algorithm to choose over 🡪 any algorithm will prefer to use as many as it can values ( is not in the vertex cover) and least values ( is in the vertex cover).

B

We will choose random vertex and run BFS in order to get the order of the tree.

We'll travel in post order and for each vertex we apply:

If any vertex has no children then and ,

Else

The optimal value is (where is the random we chose above)

Correctness

Any vertex could be in or out from the vertex cover, for each of the vertices we calculate what happens if any vertex in or out.

1. If a vertex is in then its value a(v) is in the vertex cover and we can either include or exclude each of his children, we are adding the option which adds least weight.
2. If a vertex is not in, then its value b(v) is in the vertex cover then we must add all his children into the vertex cover.

In the end we choose the least weight of the root vertex.

C - Best result is 25

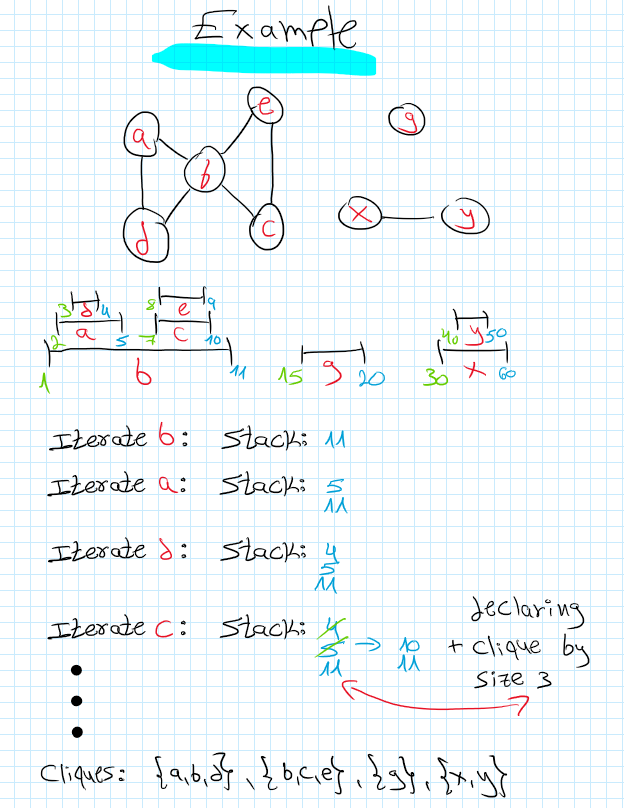
|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **a** | **b** | **c** | **d** | **e** | **f** | **g** | **h** | **i** | **j** | **k** |
| **order** | 1 | 2 | 4 | 9 | 3 | 5 | 11 | 10 | 8 | 6 | 7 |
|  | 3 | -5 | 3 | 15 | 1 | 8 | **25** | 12 | 11 | 4 | 1 |
|  | 2 | 7 | 3 | 33 | -2 | 2 | 35 | 5 | 1 | 5 | 6 |

Question 2

Algorithm

1. Build a laminar family of intervals as following:
   1. initiate
   2. travel recursively through each of the trees, start from the root, when reaching a new node (include the root of the tree) add new with and increase by . when there are no more children to travel into, set the of the node's to and again increase by .
2. Run over the (they are sorted). For each add its to a stack, if the is bigger than the first element (top element) in the stack, declare on the new clique and start to pop all the elements in the stack that smaller than the .
3. **Note -** Equivalent algorithm is to perform DFS from each of the roots to each of the leafs and declare a clique to each of the paths (root to leaf)

Overall,



Correctness

We are given a forest of rooted trees and each vertex corresponds to an interval and is an ancestor of a vertex if and only if therefore we know for sure each vertex's interval his children and their children intervals are contained in the vertex's interval. This way we can created a sorted laminar family.

Given a laminar family therefore we know either they don't intersect at all or one of them is contained in the other. The algorithm works correctly because we are creating a clique with the most vertices possible. We know the clique is with the most vertices possible because we keep adding vertices to the clique until . If then 🡪 in the same clique.

This algorithm will produce us minimum clique cover because we are creating cliques biggest as possible so there isn’t a clique that contained in another clique.

**Complexity time -** creating laminar family of intervals is and creating minimum clique cover is because over each interval we are executing commands, where .

Overall,

Question 3

Assume is a bipartite graph whose vertices can be divided into two disjoint and independent set (morning activities) and (afternoon activities). Each edge between and is an activity selection that a worker made.

Rules

1. For each and must be in the solution 🡪 add into morning activities that Oren selecting (only if ). Remove all edges associated with and decreased by one. **Justification -** If is not in the solution that’s mean we need to choose at least activities in because we need to make sure at least each worker receives one activity (Morning/Afternoon).
2. For each and must be in the solution 🡪 add into afternoon activities that Oren selecting (only if ). Remove all edges associated with and decreased by one. **Justification -** Same as rule 1 justification.
3. If return False. **Justification -** If we can’t choose more activities and there are still workers without at least one activity there is not solution.

Kernel Size

If we reached this far we know for sure that each of the vertices in the morning their degree is maximum and the vertices in the afternoon their degree is maximum therefore, in the worst case we can have edges and maximum vertices

Algorithm

1. Iterate each of the vertices that have degree bigger than 0 (1 and above), for each of them apply the rules. This cost us at maximum run though vertices and edges 🡪 complexity time is .
2. Perform brute-force on our kernel which has at maximum edges and vertices, .

Overall complexity time - .

Question 4

|  |  |  |  |
| --- | --- | --- | --- |
| **Processing time** | 100 | 1 | 1 |
| **Release time** | 12:00 | 13:00 | 13:00 |

When adding to each of the jobs this causing a "snowball" effect, the first job adding to the new solution, the second job on the same machine adding and so on…

We have machines and each of the machines handles jobs (We know that because there isn't release time and the jobs being sorted before, ).

In general case with machines, , and in our case with 2 machines,

Question 5

I read and understood

Yes, the algorithm can replace the algorithm

We know that the first jobs are scheduled optimally.

All the machines in the algorithm are busy at the time , that’s what determine the makespan 🡪 this is a property of any list scheduling without intended idle.

The starts its processing in the non increasing sorted order because of step b.