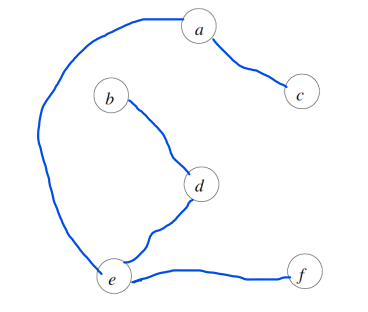
Chimzim Ogbondah

Com S 311

22 April 2021

Exam 2

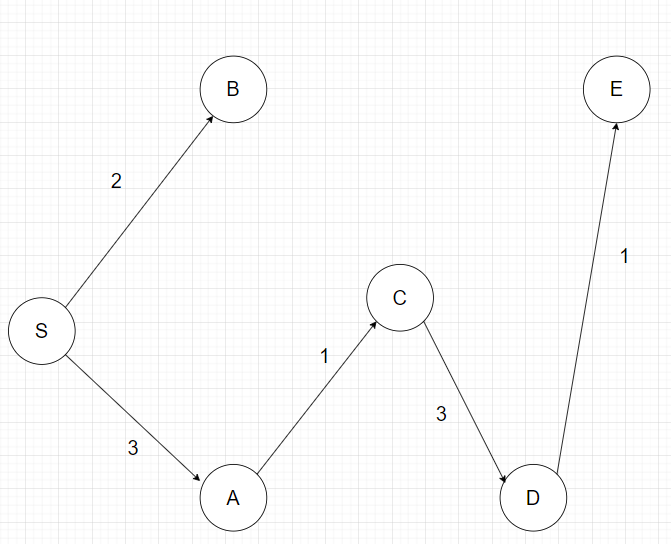
1. Minimum spanning tree problems
   1. 

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 1 | 2 | 3 | 4 | 5 |
| Kruskal’s | (A, C) | (E, F) | (B, D) | (A, E) | (E, D) |
| Prim’s | (A, C) | (A, E) | (E, F) | (E, D) | (D, B) |

* 1. Note the Starting vertex is A.
  2. Suppose there are two minimum spanning tress A & B and e is an Edge with the smallest cost that is in A but not in B. Now let us say *e* connects vertices Y-Z, B must contain a path Y-Z that is not just made up of the edge *e*. Now if we add *e* to add B it would create a cycle between Y-Z. Thus, if all other edges were in A, A would also have a cycle which it can not since we stated it is a MST. This means that there has to be an edge let us say *f* in the cycle that is not in A. Since all the edges are of different cost *f* must be greater than *e* (Because it is not in the minimum spanning tree of A). So by replacing *f* with *e,* it would give a MST with a smaller total cost. (Proof by contradiction)

1. Dijkstra

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Iteration | s | a | b | c | d | e | Selected Node |
| 0 | 0 | ∞ | ∞ | ∞ | ∞ | ∞ | s |
| 1 | 0 | 3 | 2 | 6 | ∞ | ∞ | b |
| 2 | 0 | 3 | 2 | 5 | ∞ | 11 | a |
| 3 | 0 | 3 | 2 | 4 | 10 | 11 | c |
| 4 | 0 | 3 | 2 | 4 | 7 | 11 | d |
| 5 | 0 | 3 | 2 | 4 | 7 | 8 | e |
| 6 | 0 | 3 | 2 | 4 | 7 | 8 | Complete |

* 1. 
  2. Suppose *e, f* are edges and *e, f* ∈ E with the smallest path y, z, a ∈ V. The path *c* from y-z-a is not in Vk to a certain vertex. There must contain some edge g ∈ E that is not in path of Vk that would make the path from s-a to a vertex not the shortest path. so, the cost of all the vertices from s to a would be greater than those in Vk to a vertex since we stated that every path is unique and non-negative. So, by putting a ∈ Vk*,* it would give a path a greater distance to some vertex that is connected with edge *g*.

1. Compliment Problem
   1. For the language L, let C be an Algorithm that decides it in polynomial time. We then can construct a Algorithm C’ that decides the reverse complement of L in polynomial time: C’= {0, 1} \* :

1.Run (negated) x on C’ : on the input {0,1}\* in reverse order. (Rather than starting at 1->n go from n->1)

2. If x == Empty : Accept

2.If x[i] = 1 set temp[i] = 0. If x[i] = 0 set temp[i] = 1.

3. Then run temp string from C’ on C and if C accepts, then accept otherwise reject

C’ decides the reverse complement of L in O(n) time which is polynomial. Since C runs in polynomial time, C’ is still runs in polynomial.

1. String Transformation in NP
   1. Proposition: Transformation is verified by a verification algorithm

A(x, y, k, c) : where C is the certificate which is the sequence of operations for x having n operations

1.For(each cert b: in c)

2. (String)temp[operationSize] = b(operation) on x

3.numOps += 1 (End of for loop)

4.IF (numOps > k || !temp == y) : reject (A(x, y, k, c) = 0 )

5.Else: accept ( A(x, y, k, c) = 1)

L = {e ∈ {0, 1}\* certificate in c s.t |c| = O(|n|C) and A(x, y, k, c) = 1 }

A is polynomial because it runs in O(n) time and size of |c| = O(|n|c) s.t. is in NP

1. Restricted CNF-SAT is in P
   1. Let L be a language that decides RES-CNF

L: takes in input θ which is a restricted Boolean formula

Alg:

For(I = 0; I < θ.size(); I++) //Iterating through θ where I is the position in the formula

If(I == ‘(‘) : inClause = **TRUE; continue;**

If(I == ‘)’ ): inClause = **FALSE; continue;**

If(I == ‘∨’) continue;

For(j = 0; j < θ.size(); j++) //iterating through θ where j is the position in the formula

If( (j != ‘V’|| j != ‘∧’ || j != ‘)’ || j != ‘(‘ ) && inClause == **TRUE** )

If(isValid != 1): continue

Else:

If( !I == j) : isValid == 0 //note is valid starts at 1 and assuming all literals negated or not are in the same position and can be check with ! function

End J loop

End I loop

If(isVaid == 1) : **Accept**

Else : **Reject**

This Algorithm is polynomial because it runs in O(n2) time where n is the size of the Boolean formula θ and it either Accepts or Rejects whether the formula is valid or not.

Assumption that each literal can be treated as an object so the you don’t have to differ between x1,x2…..xn