Name : Omkar Dixit

GWID : G42396299

Home Work IV



Formulation of problem:

* Every digraph of n nodes is represented by a 2D array A[1:p+q,1:p+q], which is the well-known adjacency matrix.
* The values of the entries in the array are binary and also are independent of one another.
* The 2D array can be represented by a a 1D binary array X[1:N] where N=(p+q)2
* The value of each X[i] is in {0,1}

Adjacency matrix:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **1** | **2** | **3** | **.** | **.** | **.** | **.** | **p** | **p+1** | **P+2** | **.** | **.** | **.** | **.** | **p+q** |
| **1** |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| **2** |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| **3** |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| **.** |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| **.** |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| **.** |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| **.** |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| **p** |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| **p+1** |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| **p+2** |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| **.** |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| **.** |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| **.** |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| **.** |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| **p+q** |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

But as it’s a bipartite graph, elements [1:p,1:p] and [p+1:p+1,p+1:p+q] will be 0 and since it’s undirected graph, X[I,j] = X[j,i].

Thus adjacency matrix can be reduced to,

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **p +1** | **p+2** | **p+ 3** | **.** | **.** | **.** | **.** | **p +q** |
| **1** |  |  |  |  |  |  |  |  |
| **2** |  |  |  |  |  |  |  |  |
| **3** |  |  |  |  |  |  |  |  |
| **.** |  |  |  |  |  |  |  |  |
| **.** |  |  |  |  |  |  |  |  |
| **.** |  |  |  |  |  |  |  |  |
| **.** |  |  |  |  |  |  |  |  |
| **p** |  |  |  |  |  |  |  |  |

Where

* The 2D array can be represented by a a 1D binary array X[1:N] where N=p\*q
* The value of each X[i] is in {0,1}

2D to 1D formula:

Constraints:

* It shouldn’t be cyclic ie,

 X[i] != X[j] for all i != j

* Degree of the node is at most K.

We will introduce new array Y consisting of degree. It will be incremented by 1 if it is included in graph and,

Complete solution:

Bound Function:

* Bound function

Function Bound(X[1:n],L,Y[1:n],k,E)

begin

/\* X[1:L-1] are assigned C-compliant

values. This function checks to

see if X[L] is C-compliant \*/

Y[L]=Y[L] + 1; //add degree of incoming node

Y[E]=Y[E] + 1; //add degree of other end of node

if Y[L] > K or Y[E]>K // check degree by adding into graph.

Y[E]=Y[E] – 1 //do not add this edge.

Y[L]=Y[L] – 1;

return (false);

endfor

1. A

Formulation of problem:

Every n bit k-localized solution can be represented as X(1,2,….,n)

X[i] takes value from 1:n

Constraints:

1. Number cannot be repeated ie,

X[i] != X[j] for all i != j

Complete solution:

1. A0 = 0
2. Bound Function :

Function Bound(X[1:n],l,k)

begin

/\* X[1:l-1] are assigned C-compliant

values. This function checks to

see if X[l] is C-compliant \*/

for i=1 to l-1 do

if X[l] = X[i] then

return(false);

endif

endfor

if(

then

return(false);

endif

return(true);

end

2b.

Formulation of problem:

Input is graph G(1,n1) and G2(1,n2). N1= number of nodes in G1 and n2 are the number of nodes in G2.

X[I] takes value from 1 to n1 and 1 to n2.

Constraints:

If I,j is and edge in G1 then f(i),f(j) should be an edge in G2

Complete solution:

A0 = 0

1. Bound Function :

Function Bound(int x[1…n],l)

Begin

For i=1 to r

Do

If(G1[i][r] == 1) && G2{x[i],x[r] != 0 )

Then

Return false

End if

done

return true

End

3 A

Input: Graph (adjacency matrix) and cost of colours

|  |  |  |  |
| --- | --- | --- | --- |
| **1** | **2** | **.** | **n** |
| **2** |  |  |  |
| **.** |  |  |  |
| **n** |  |  |  |

Finding permutation that

Output:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Node | 1 | 2 | …… | n |
| Colour No | 1/2/3 | 1/2/3 | …… | 1/2/3 |

Output array is from [1:n] and value can be any colour (1 or 2 or 3).

Constraints for selection:

1. Same colour cannot be selected for adjacent points.

Derivation of cost function:

Assign possible minimum value to all nodes except adjacent nodes.

Definition of cost function:

C^= cost so far + provided that Wi !=Wk if I,k are adjacent.

Proof:

Let E be the E-node that happens to be a final leaf.

Need to prove that C(E) <= C(X) for any live node X.

C(E)=C^(E) because E is a final leaf

C^(E) <= CC(X) for any live node X, because E is the expanding node, that is, the minimum-C^node

C^(X) <= C(X) by assumption.

Therefore, C(E)=CC(E)<=C^(X) <=C(X), leading to C(E)<=C(X).

Hence proved

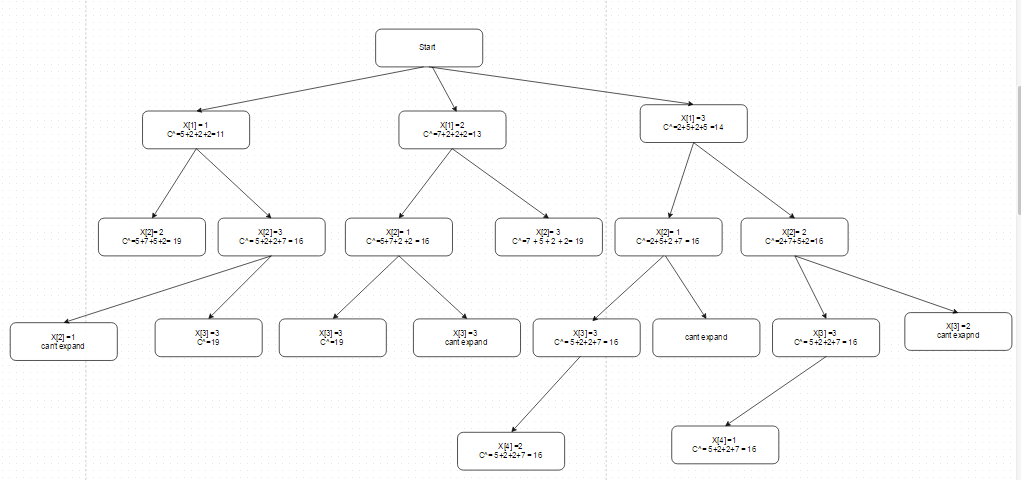
3 b. Input graph:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 1 | 2 | 3 | 4 |
| 1 | 0 | 1 | inf | 1 |
| 2 | 1 | 0 | 1 | 1 |
| 3 | inf | 1 | 0 | 1 |
| 4 | 1 | 1 | 1 | 0 |

Colour values:

|  |  |  |
| --- | --- | --- |
| 1 | 2 | 3 |
| 5 | 7 | 2 |

Applying C^ function :



C^ function:

C^= cost so far + provided that Wi !=Wk if I,k are adjacent.

Applying and expanding tree :

Solution: (as tree is symmetric)

Solution 1:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Node | 1 | 2 | 3 | 4 |  |
| Color no | 3 | 2 | 3 | 1 |  |
| Weight | 2 | 7 | 2 | 5 | **Total = 16** |

Solution 2:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Node | 1 | 2 | 3 | 4 |  |
| Color no | 3 | 1 | 3 | 2 |  |
| Weight | 2 | 5 | 2 | 7 | **Total = 16** |

4.

Input: Graph (adjacency matrix) and cost of colours

|  |  |  |  |
| --- | --- | --- | --- |
| **1** | **2** | **.** | **n** |
| **2** |  |  |  |
| **.** |  |  |  |
| **n** |  |  |  |

Output Nodes : [ 1 to N ,1 ]

Constraints for selection:

1. Each node should be visited once.

2. Last node should be connected to 1st node.

Derivation of cost function:

Assign possible minimum value to all nodes except adjacent nodes.

Definition of cost function:

C^= cost so far + if k is not visited.

Proof:

Let E be the E-node that happens to be a final leaf.

Need to prove that C(E) <= C(X) for any live node X.

C(E)=C^(E) because E is a final leaf

C^(E) <= CC(X) for any live node X, because E is the expanding node, that is, the minimum-C^node

C^(X) <= C(X) by assumption.

Therefore, C(E)=CC(E)<=C^(X) <=C(X), leading to C(E)<=C(X).

Hence proved

5.

a.

Divide and conquer algorithm:

Procedure find\_hamiltonian (input A[1:nodes], i,j; output B[1:n])

begin

Datatype C[1:n];

If i=j then B[i] = A[i]; Return; endif

Mergesort (A,i,(i+j)/2;C); /\* sorts the first half\*/

Mergesort(A,(i+j)/2 +1,j;C); /\* sorts the second half\*/

Merge(C,i,j;B); /\* merges the two sorted halves \*

\* into a single sorted list \*/

End

Join last element to 1st element

Procedure Merge(input: C i,j; output: B)

begin

int k=(i+j)/2;

int u,v,w; /\* u will scan C[i:k],

v will scan C[k+1:j], and

w will index the out B\*/

u=i;

v=k+1;

w=u;

while (u <= k and v <= j) do

check all edges from u to v and sign minimum edge to array. // for joining 1,23 and 4,5,6 then if 1-4 is minimum then output will be 4,1,5,6 and so on

endwhile

If u > k then

Put C[v:j] in B[w:j];

Elseif v>j

Put C[u:k] in B[w:j];

endif

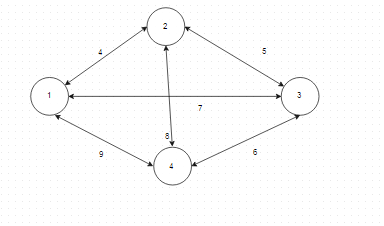
end

Time Complexity = T(n) = 2T(n/2) + cn

T(n) = o(nlogn)

b.

Consider graph



As per greddy

c. Greedy algorithm:

It will Start from node 1 and will check minimum weight edge and will go to that point and will remove

Procedure find\_hamiltonian(in:nodes, E[1:n];out:G)

begin

Put in T the n nodes and no edges;

while T has no nodes do

Choose a remaining edge e from node x of

minimum weight reaching to node Y;

Delete x from the graph;

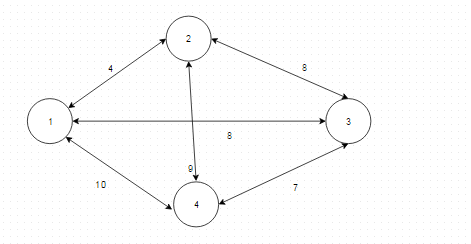
proceed graph to Y.

Add e to T;

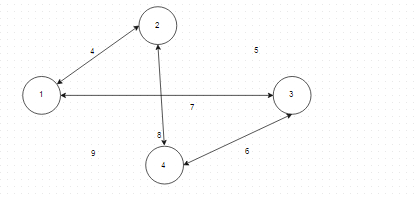
endwhile

end find\_hamiltonian

d. Consider example,

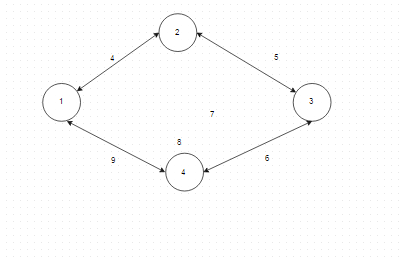


Divide and conquer output:



Hamiltonian cycle = 1-2-4-3-1. Weight = 25.

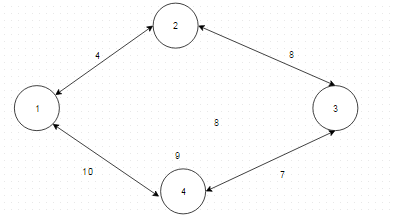
Better Hamiltonian path: 1-2-3-4-1. Weight =24.



Greedy cycle:

Start at node 1 then will go to node 2 then to 3 then to 4 and then back to 1.

1-2-3-4-1 -----------------------------------------🡪 Weight = 4+8+7+10=29



Better Hamiltonian cycle :

1-2-4-3-1

Cost = 4+8+7+9 = 28.

