Algorithm Analysis and Design

Assignment - 5

Aryaman Kolhe
2022121002

I Basic Graph Algorithms

Q1. A graph (V, E) is bipartite if the vertices V can be partitioned into two subsets L and R $\Rightarrow \forall e \in E \quad \exists \; v \in L, u \in R$ where $e = (v, u)$

(a) Prove that every tree is bipartite.

PROOF

[1] Assume that there exists a tree $T(V, E)$ which is not bipartite.

There is only one path from . Assign a vertex v to be in set R.

The BFS tree on V will have only adjacent vertices u_i to v on the next level. All u_i in level L_1 will be in set L.

Adjacent vertices cannot be in the same set (either L or R).

2 paths from the root to any vertex will imply that there is a cycle in the tree, which is not possible. Since T is a tree, BFS will not revisit a vertex, so the L, R classification will happen without conflicts.

Q.E.D.

(b) Prove that a graph G_1 is bipartite iff every cycle in G_1 has an even number of edges.

PROOF BY CONTRADICTION - Part (1)

Assume that G_1 is bipartite with at least one odd cycle.

Assume that $\exists$ cycle $= v_1, v_2, \ldots, v_k, v_1$.

Assume without loss of generality that $v_1 \in L$.

By definition of bipartiteness, since $(v_1, v_2) \in E$, $v_2 \in R$.

Similarly, $v_3 \in L$, $v_4 \in R$

For i = Even, the node is in R.

For i = odd, the node is in L

For the odd cycle, to end, the last node is v_1, which implies that $v_1 \in R$. [contradiction].

~~Second~~ ~~direction~~:

Case 1 : $v_1 \in R$

case 2 : $(v_k, v_1) \in E(G_1)$, $v_k, v_1 \in L$.

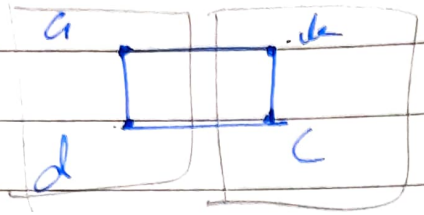
~~Both~~ Both are contradictions.

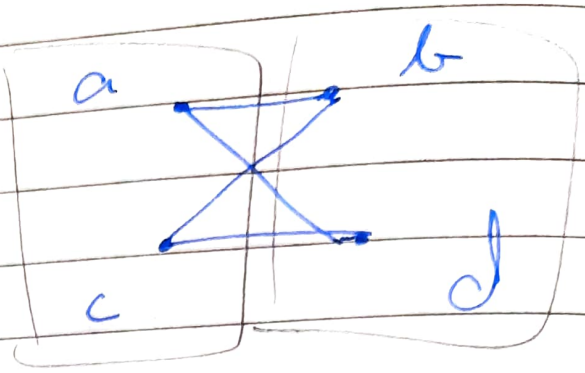
∴ If a graph is bipartite, it has no odd cycles.

Part (2) - Prove that if a graph has no odd cycles, it is bipartite.

Assume: The only cycles are even cycles and ∃ edges between pairs of vertices in the same part.

Base case





We can re-assign the partition to make G bipartite.

~~Testing~~ Bipartiteness using BFS.

G is connected

Start with a vertex S

$L_0 = \{s\}$

$L_1, \ldots, L_k$.

$(L_0 \cup L_2 \cup \ldots, L_k)$ are at an even distance from each other.

Same for $(L_1 \cup L_3 \cup \ldots \cup L_{k-1})$

Let there be an edge between u and v ∋ $u \in L_i$, $v \in L_3$ [WLOG]

Shortest distance of u from s is 1.

Shortest distance from v and s is 3.

If $\exists (u,v) \in E(G)$ then the minimum distance between u and s should have been 2 or 4 (since dist $\min(v,s) = 3$) [derived from the BFS tree.] (contradiction.) $\therefore$ Q.E.D

(c) Algo: to test for Bipartiteness using BFS.

→ Run BFS
→ Club even layers into A
→ Club odd layers into B.

if there are edges in G b/w pairs of vertices in either A or B, return FAIL ; else return (B, B, E).

Testing for connected components:
Use BFS(v_1). If the queue becomes empty, loop through visited[ ] to find a node x that hasn't been visited. Then do BFS(x). Repeat until all i in visited[ ] are explored.

If G is disconnected, after getting A,B from test for bipartiteness and if A' and B' are also connected components, you can make 2 bipartite graphs.

(A ∪ B', B ∪ B')
OR (A ∪ B', B ∪ A')

Time : $O(m+n)$
Since it is done with BFS.

5

V Network Flows:

1. A given flow network G may have more than one minimum (s-t) cut, let us define the best minimum (s,t) cut to be any minimum cut [S, T] with the smallest number of edges crossing from S to T. Describe an algo to find the best min (s,t) cut when the capacities are integers.

By finding minimum flow f.

Algorithm 0. Create an array A.

1. Find a minimum (s,t) cut M. using the procedure described at the end. Record len(M) in A.
2. Increase the capacity of any edge in M by 1.

Now, if the min cut was unique in G, the the max flow of G would have changed

3. Find the max flow of the new graph = f'

If f' ≠ f the min cut is unique
If f = f the min cut is not unique (∃ another min cut which gave f' = f)

4. If there is a new min cut, then find it using 1.
Record len(Mi) in A.
A.push(Mi).

5. Keep doing this till the condition in 3. does not show that there is a remaining min cut.

6. Loop through A to get M_{min} (the minimum value) in $O(n)$.

Return $M_{min} = \min_j \{ j \in M \text{ and } M \text{ is min} \}$

Procedure to find min cut -

1. Run Ford Fulkerson algo in polynomial time $O(|f^*|E)$, and consider the final residual graph G_R.
2. Use DFS to find the set of vertices that are reachable from the source in G_R $= m \cdot O(m+n)$
3. Let this set of vertices be A.

Let $B = V - A$.

Then (A, B) is a min cut of G.

Time: $O(m(|f^*|E) + m^2(m+n) + m_{min})$ (upper bound for num min cuts)

$=$ Polynomial time.

II. Greedy Algorithms:

2. A spanning tree T of an undirected positively weighted graph G is called a minimum bottleneck MST (MBST) if the edge with max. cost is minimum among all possible spanning trees. Show that an MST is always a MBST. What about the converse.

Suppose that T is a minimum spanning tree of G, and T' is a spanning with a lighter bottleneck edge.

∴ ∃ edge $e \in E(T)$

such that

e is heavier than every edge in T'.

∴ if we add e to T', it will create a cycle C on which it is the heaviest edge (since all other edges in C belong to T').

By the Cut Property; e does not belong to any minimum spanning tree, contradicting the fact that it is in T, and T is an MST.

The converse is not true.
Let G have vertices $\{v_1, v_2, v_3, v_4\}$ with edges between each pair of vertices and with weight on edge from v_i to v_j equal to $i+j$.

Then, every tree has a bottleneck edge of weight at least 5. The tree consisting of a path through vertices $v_3\ v_2\ v_1\ v_4$ is a minimum bottleneck tree.
It is not an MST, ∵ total weight > tree weight with edges from v_1 to every other vertex

QED

III. Divide and Conquer

3. Describe and analyze a variant of Karatsuba's algo that multiplies any m-digit number and any n digit number, for $n \geq m$ in $O(n \cdot m^{\log_2 3 - 1})$.

Algorithm
Split the larger number into $\lceil n/m \rceil$ chunks, each with m digits. Multiply the smaller number by each chunk in $O(m^{\log_2 3})$ time using Karatsuba's algorithm.

Then add the resulting partial products with appropriate shifts.

9a

```
skew multiply ( x[0 ... m-1], y[0 ... n-1] ):
    prod = 0
    offset = 0
    for i = 0 to ⌈n/m⌉ - 1
        chunk = y[i.m ... (i+1).m - 1]
        prod = prod + multiply(x, chunk) . 10^(i.m)
    return prod.
```

Analysis:

Each call to multiply requires $\Theta(m^{\log 3})$ time and all other work within a single iteration of the main loop requires $\Theta(m)$ time.

∴ Overall running time of the algo is:

$$\Theta(1) + \lceil n/m \rceil . \Theta(m^{\log 3 - 1} n)$$

// o.t.p.

Note that this is the standard method for multiplying a large integer by a single digit integer written in base 10^m but with each single-digit multiplication implemented using karatsuba's algorithm

@10

Recursion and Dynamic Programming:

4. Largest independant set ≡ NP Hard.
But in some special classes of graphs, we can find the largest independent sets quickly
In particular, when $G = Tree(V, E)$
$|V| = n$, we can compute the largest independent set in $O(n)$.

Let $LIS(n)$ indicate the size of largest independant set of a tree with root x.

$$LIS(x) = Max \{ (1 + \text{sum of LIS for all grandchildren of } x), (\text{sum of LIS } \forall \text{ children of } x) \}$$

OR.

$$LIS(x) = max \left\{ 1 + \sum_{w: \text{grandchildren of } x} LIS(w), \sum_{w: \text{children of } x} LIS(w) \right\}$$

done in $O(m+n)$

The recursion makes sense as a node which is a part of the LIS, then its children cannot be part of LIS, but its grand children can be. (same "choosing" argument from DP.)

Q11

We can build an array of size i-1 and from there we can calculate LIS(i)
No. of unique sub problems = O(n)

∴ Overal time = O(n+m)

VI. P vs NP

1. Show that Sudoku is in NP.

Ans: ~~Overhead~~ Idea -
Given a solution to $N^2 \times N^2$ Sudoku, we can check if it's valid or not by checking if the following three conditions are met.

→ Each row contains unique values from 1 to N^2. Each column contains unique values from 1 to N^2.
→ Each of the N^2 sub-squares, of size $N \times N$ contains a unique value from 1 to N^2
→ All this can be done in $O(N^c)$ $c \in R^+$

∴ Sudoku is in NP

OR -
We need to create a verifier in P-time.

PTO.

for $n=3$ we get the normal sudoku problem.

We can verify by checking 3 conditions.

1. Each row has unique values 1, 2, ... n.
   Check in $O(n^2)$ for 2 for loops nested
2. Same for columns. $O(n^2)$
3. For subsquares, we can use a sliding window of size $n \times n$ in $O(n^3)$.

∴ Verifier takes $O(n^3)$ time

∴ Π is in NP.

// QED.

References.

I. Class Notes

V. Assignment 4 referred (to find mine cut)

III. Courses.engr.illinois.edu