CSCI 5870: Data Structures and Algorithms
Homework 4

Instructor: Dr. Bob Kramer

Rabin Thapa

1. [Optimal Matrix Multiplication]
   Calculate the optimal number of scalar multiplications necessary to calculate the matrix product $M = M_1 \cdot M_2 \cdot M_3 \cdot M_4 \cdot M_5$, where M_1 is a 4×3 matrix, M_2 is a 3×5 matrix, M_3 is a 5×3 matrix, M_4 is a 3×2 matrix and M_5 is a 2×4 matrix. Show both the cost and "how" matrices ("how" shows the choice made for each $f(i, j)$ value that yields the minimum value.)

i , j	1	2	3	4	5
1	0	60	81	84	116
2	-	0	45	60	84
3	-	-	0	30	70
4	-	-	-	0	24
5	-	-	-	-	0

Figure 1: f-table

i , j	1	2	3	4	5
1	-1	1	1	1	4
2	-	-1	2	2	4
3	-	-	-1	3	4
4	-	-	-	-1	4
5	-	-	-	-	-1

Figure 2: k-table

Calculations:
$f(1,2) = 4 \cdot 3 \cdot 5 = 60$,
$f(2,3) = 3 \cdot 5 \cdot 3 = 45$,
$f(3,4) = 5 \cdot 3 \cdot 2 = 30$,
$f(4,5) = 3 \cdot 2 \cdot 4 = 24$,
$f(1,3) = min\{f(1,2) + 60, f(2,3) + 36\} = min\{120, 81\} = 81$,
$f(2,4) = min\{f(2,3) + 18, f(3,4) + 30\} = min\{63, 60\} = 63$,
$f(3,5) = min\{f(3,4) + 40, f(4,5) + 60\} = min\{70, 104\} = 70$,
$f(1,4) = min\{f(1,1) + f(2,4) + 4 \cdot 3 \cdot 2, f(1,2) + f(3,4) + 4 \cdot 5 \cdot 2, f(1,3) + f(4,4) + 4 \cdot 3 \cdot 2\} = min\{84, 130, 105\} = 84$,
$f(2,5) = min\{f(2,2) + f(3,5) + 3 \cdot 4 \cdot 4, f(2,3) + f(4,5) + 3 \cdot 3 \cdot 4, f(2,4) + f(5,5) + 3 \cdot 2 \cdot 4\} = min\{130, 105, 84\} = 84$,
$f(1,5) = min\{f(1,1) + f(2,5) + 4 \cdot 3 \cdot 4, f(1,2) + f(3,5) + 4 \cdot 5 \cdot 4, f(1,3) + f(4,5) + 4 \cdot 3 \cdot 4, f(1,4) + f(5,5) + 4 \cdot 3 \cdot 4\} = min\{132, 220, 153, 116\} = 116$.

So, the algebraic form of optimal matrix multiplication is $(M_1(M_2(M_3M_4)))M_5$.

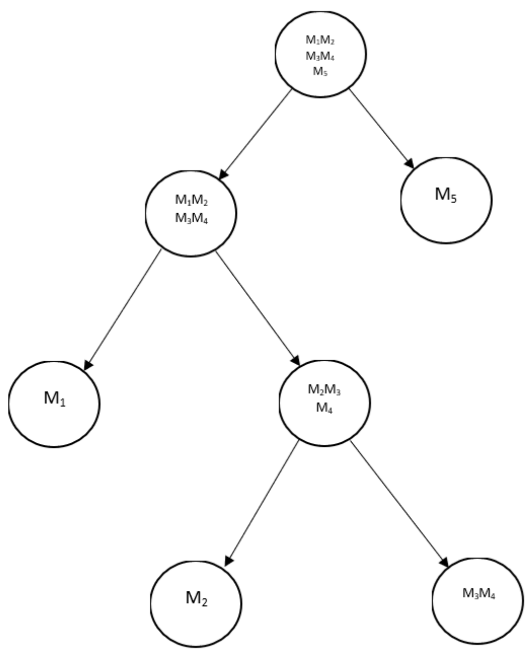

Figure 3: Optimal Matrix Multiplication Tree

2. [Optimal Binary Search Tree]
Construct an optimal binary search tree for the six nodes A, B, C, D, E and F, where the nodes have the following probabilities:

n	$P(n)$
A	0.20
B	0.08
C	0.17
D	0.15
E	0.25
F	0.15

Show both the cost table and the "how" table.

A to B:

i	j	k	$S(i,j)$	Cost
1	2	1	28	36
1	2	2	28	48

B to C:

i	j	k	$S(i,j)$	Cost
2	3	2	25	42
2	3	3	25	33

C to D:

i	j	k	$S(i,j)$	Cost
3	4	3	32	47
3	4	4	32	49

D to E:

i	j	k	$S(i,j)$	Cost
4	5	4	40	65
4	5	5	40	55

E to F:

i	j	k	$S(i,j)$	Cost
5	6	5	40	65
5	6	6	40	55

A to C:

i	j	k	$S(i,j)$	Cost
1	3	1	45	78
1	3	2	45	82
1	3	3	45	81

B to D:

i	j	k	$S(i,j)$	Cost
2	4	2	40	87
2	4	3	40	63
2	4	4	40	73

C to E:

i	j	k	$S(i,j)$	Cost
3	5	3	57	112
3	5	4	57	99
3	5	5	57	104

D to F:

i	j	k	$S(i,j)$	Cost
4	6	4	55	110
4	6	5	55	85
4	6	6	55	110

A to D:

i	j	k	$S(i,j)$	Cost
1	4	1	60	123
1	4	2	60	127
1	4	3	60	111
1	4	4	60	135

B to E:

i	j	k	$S(i,j)$	Cost
2	5	2	65	164
2	5	3	65	128
2	5	4	65	123
2	5	5	65	128

C to F:

i	j	k	$S(i,j)$	Cost
3	6	3	72	157
3	6	4	72	144
3	6	5	72	134
3	6	6	72	171

A to E:

i	j	k	$S(i,j)$	Cost
1	5	1	85	208
1	5	2	85	204
1	5	3	85	176
1	5	4	85	185
1	5	5	85	196

B to F:

i	j	k	$S(i,j)$	Cost
2	6	2	80	214
2	6	3	80	173
2	6	4	80	158
2	6	5	80	158
2	6	6	80	203

A to F:

i	j	k	$S(i,j)$	Cost
1	6	1	100	273
1	6	2	100	281
1	6	3	100	221
1	6	4	100	233
1	6	5	100	226
1	6	6	100	222

f-table

	0	1	2	3	4	5	6
1	0	20	36	78	111	176	221
2	-	0	8	33	63	123	158
3	-	-	0	17	47	99	134
4	-	-	-	0	15	55	85
5	-	-	-	-	0	25	55
6	-	-	-	-	-	0	15
6	-	-	-	-	-	-	0

k-table

	0	1	2	3	4	5	6
1	-1	1	1	1	3	3	3
2	-	-1	2	2	3	4	5
3	-	-	-1	3	3	4	5
4	-	-	-	-1	4	5	5
5	-	-	-	-	-1	5	5
6	-	-	-	-	-	-1	6
6	-	-	-	-	-	-	-1

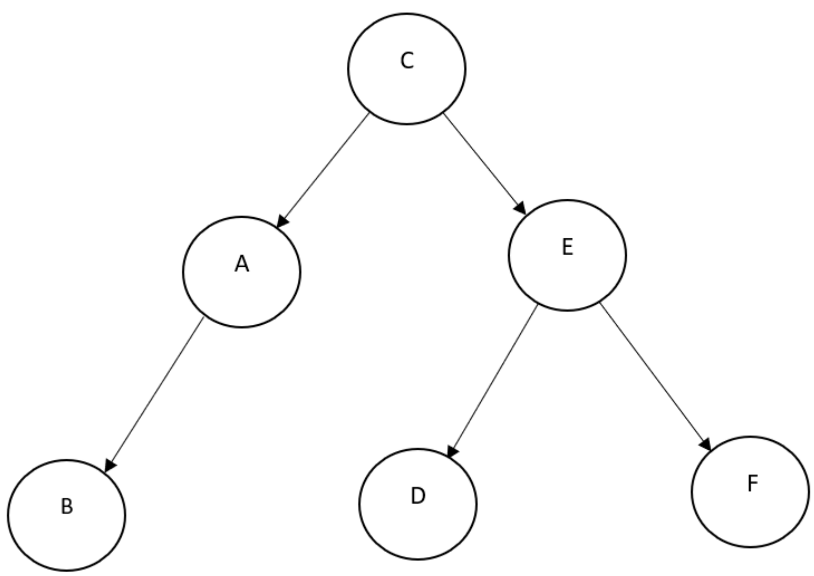

Figure 4: Optimal Binary Tree

3. [NP-Completeness #1]
A graph $G =< V, E >$ has a *Hamiltonian cycle* if there is some permutation of the vertices $\pi =< \pi_1, \pi_2, \cdots, \pi_V >$ such that there is an edge between v_{π_i} and $v_{\pi_{i+1}}$ and an edge between v_{π_1} and v_{π_V}. In simpler terms, you can start at any vertex, follow a sequence of edges to visit every vertex once, and end up at your starting vertex.

Show that the problem of determining whether or not an arbitrary undirected graph has a Hamiltonian cycle is $\mathcal{NP}$-complete.

Note: There are at least two ways to do this. One is definitely easier than the other.

Hamiltonian Cycle problem is in NP: A problem is in NP if a proposed solution for the problem can be verified in polynomial time. For a graph $G(V, E)$ with n vertices, this can be done by checking if the proposed solution (sequence of vertices) contains all n vertices of G and if they form a cycle, which can be done in polynomial time:

Preconditions: A sequence (directed set of vertices) $S = (v_0, v_1, ..., v_n)$
Postconditions: returns $true$ if S is a Hamiltonian cycle of G, $false$ otherwise

H-CYCLE(S, n)
$result \leftarrow true$
if (S contains n nodes and there is an edge (v_0, v_{n-1}))
  for $i = 0$ to $n - 1$
    if there is an edge (v_i, v_{i+1})
      continue
    else
      $result = false$
      break
  end for
else
  $result = false$
return $result$

Hamiltonian Cycle problem is NP-hard: It can be shown by reducing Hamiltonian path problem, which is NP-hard, into this problem. That means for a graph $G(V, E)$, the Hamiltonian path problem can be reduced into finding whether or not a graph $G'(V', E')$ has a Hamiltonian cycle. $G'(V', E')$ can be constructed by adding edges from a new vertex v_n to each vertex in the graph G, so $|V'| = |V| + 1$ and $|E'| = |E| + |V|$. This can be done in polynomial time. Now,
If the graph G' contains a Hamiltonian cycle C (which includes vertex v_n), then $G = G' - v$ contains a Hamiltonian path $C - v_n$.
If the graph G contains a Hamiltonian path L, then $L + v_n$ forms a cycle (since there are edges joining end vertices of L and v_n) which is a Hamiltonian cycle for G'.
So, G has a Hamiltonian path if and only if G' has a Hamiltonian cycle. This shows that a Hamiltonian cycle problem can be reduced into a Hamiltonian path problem. Hence, Hamiltonian Cycle problem is NP-hard.

[Resource:https://www.geeksforgeeks.org/proof-that-hamiltonian-cycle-is-np-complete]

4. [NP-Completeness #2]
A *clique* of a graph $G =< V, E >$ is a subgraph of G such that the subgraph is completely connected; *i.e.*, any two vertices in the subgraph are connected by an edge.

The *clique cover* problem takes a graph G and an integer k and determines whether or not the vertices of G can be partitioned into exactly k cliques.

Show that clique cover is $\mathcal{NP}$-complete.

Clique cover problem is in NP: A solution for Clique cover problem given by a non-deterministic algorithm can be verified using a deterministic algorithm in polynomial time.

Preconditions: A set of partitions $C = C_1, C_2, ..., C_k$ of vertices of G
Postconditions: returns $true$ if C is a k clique cover of G, $false$ otherwise

CLIQUE(C, k)
$result \leftarrow true$
for $i = 1$ to k
    if COMPLETE(C_i) $= true$ //checks if C_i a complete subgraph in $O(n^2)$ time
        continue
    else
        $result \leftarrow 0$
end for
return $result$

Clique cover problem is NP-hard: It can be shown by reducing k-coloring problem, which is NP-hard, into this problem. Let consider a graph $G(V, E)$ and its complement G'. Now,
If a k-coloring of G, vertices with same colors are not adjacent to one another. But in G', there is an edge between each pair of same-coloured vertices, so they form a clique, and there of k such cliques. So, if G has k-colouring, then G' has a k cliques cover.
If G' can be partitioned into exactly k cliques, then vertices in the same clique can be coloured the same and different cliques be colored different to result in k-coloring of G.
Hence, a graph G has k-coloring if and only if G' can be partitioned into exactly k cliques. So, Clique cover problem is NP-hard.

[Resource:https://en.wikipedia.org/wiki/Clique_cover]