Clique Cover Problem

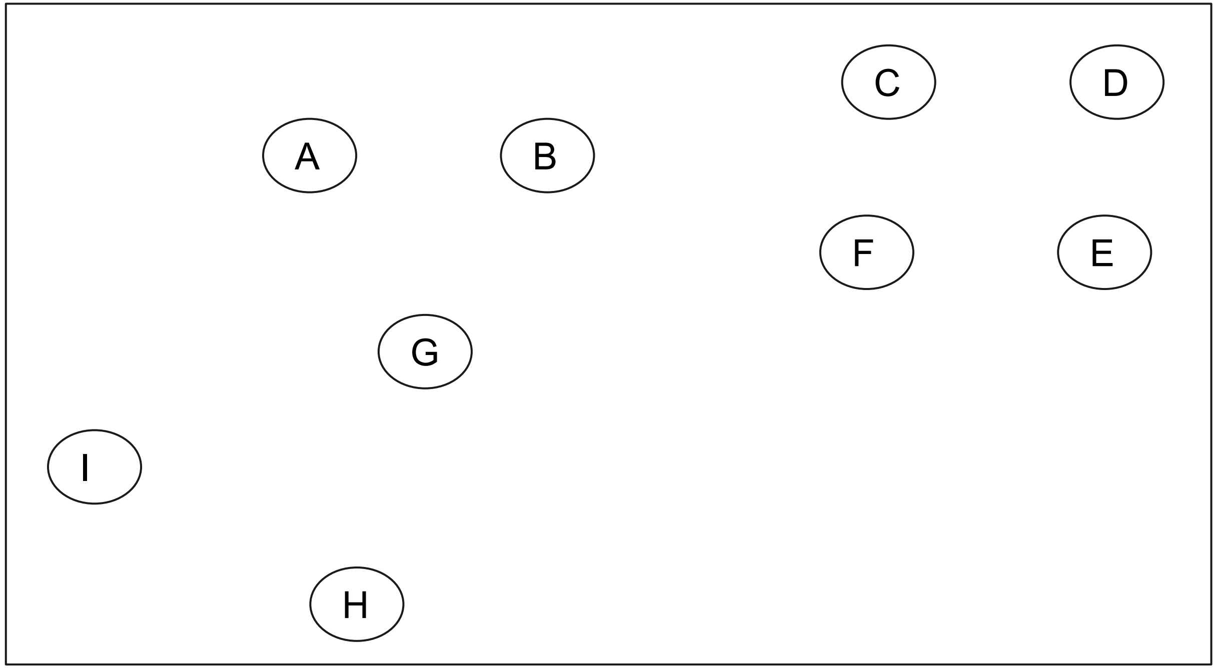
Brute Force and Exact Algorithm

Algorithm for Restricted Class of Graph

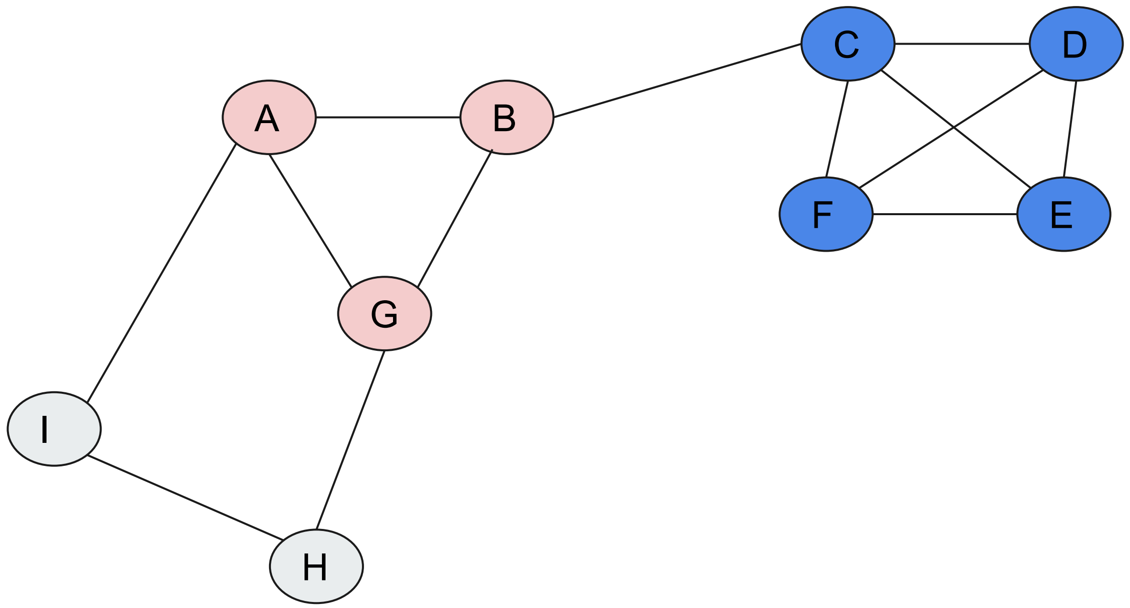
Group No: 10

Student ID:
1505002
1505044
1505057
1505097
1505101

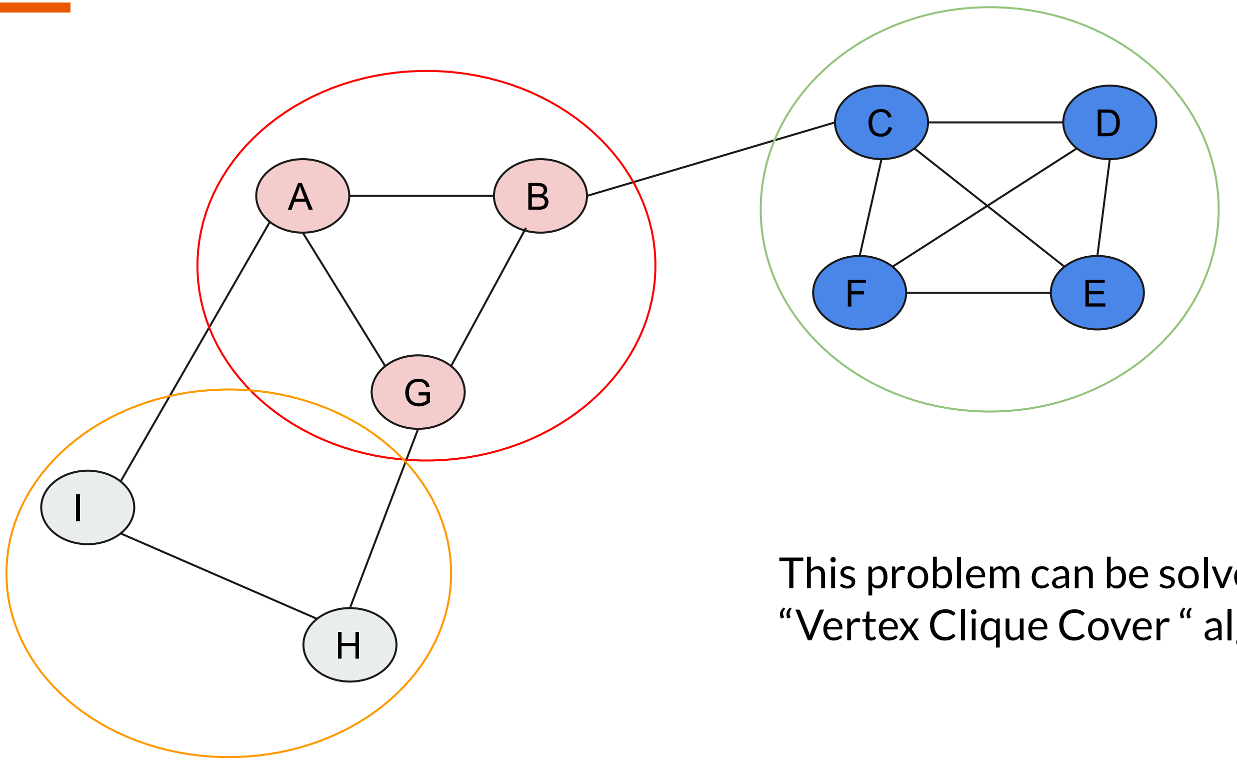
Let’s look at a scenario...



Let’s look at a scenario...



Let’s look at a scenario...



This problem can be solved using “Vertex Clique Cover “ algorithm.

Vertex Clique Cover

Clique Cover Decision Problem:

Given a graph G(V, E) and a number K, we need to answer yes or no if we can partition V(G) in K cliques

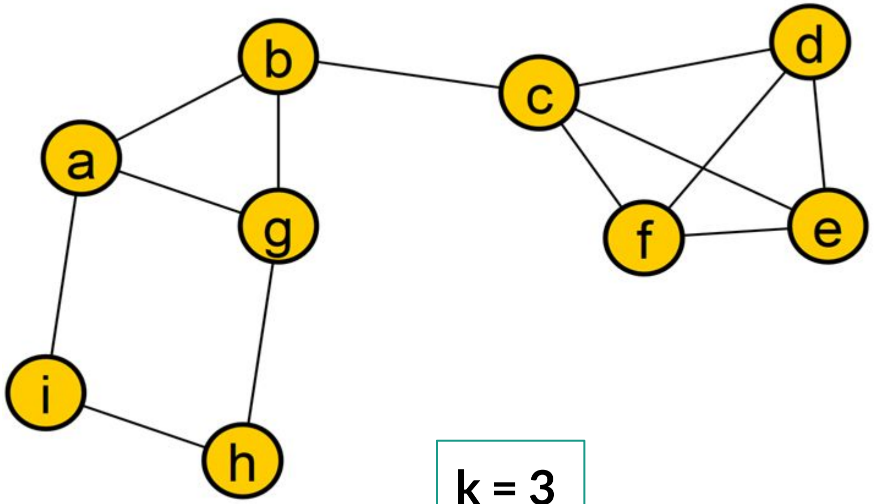
Minimum Clique Cover Problem:

Given a graph G(V, E), we need to output the smallest number for which clique cover exists. This number is called the clique cover number.

In this presentation we will discuss some algorithms regarding these problems. As already discussed, we can derive a solution for clique cover decision problem from clique cover number and vice versa.

Brute Force Algorithm

Step 1 : Find the power set of **V** where **V** is the set of vertices of graph **G**.
Step 2 : Take an empty list **S**.
Step 3 : Take every element(which is actually a subset of V) of **P(V)** and check whether it's a complete subgraph or clique then add this to list **S** otherwise skip the element.
Step 4 : Find all possible subsets of **S** of size **k**.
Step 5 : Take every subsets and check whether it covers all vertices of **V** or not.



k = 3

PowerSet, P(V)

V = {a,b,c,d,e,f,g,h,i}

P(V) =

{

{a,b,c,d,e,f,g,h,i},

……………………..

{a,b,c,g},

{c,d,e,f},

…………..

{a,b,g},

{a,b,i},

{c,e,f},

{c,e,d},

{c,d,f},

{e,d,f},

……….

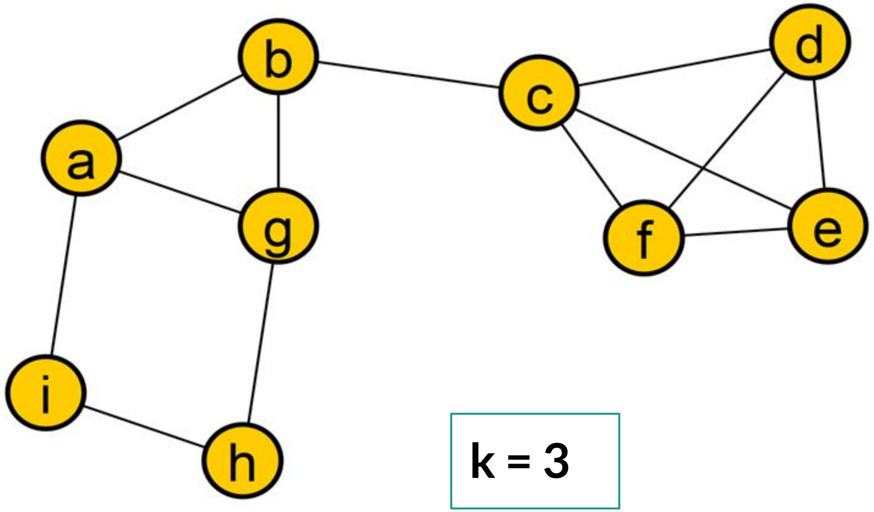
{a,b},{a,g},{b,g},{a,i},

{g,i},{i,h},{g,h},{b,c},{c,d},

{c,f},{e,f},{d,e},{c,e},{f,d},

{a},{b},{c},…………

}



k = 3

PowerSet, P(V)

V = {a,b,c,d,e,f,g,h,i}

P(V) =

{

{a,b,c,d,e,f,g,h,i},

……………………..

{a,b,c,g},

{c,d,e,f},

…………..

{a,b,g},

{a,b,i},

{c,e,f},

{c,e,d},

{c,d,f},

{e,d,f},

……….

{a,b},{a,g},{b,g},{a,i},

{g,i},**{i,h},{g,h},{b,c},{c,d},**

{c,f},{e,f},{d,e},{c,e},{f,d},

{a},{b},{c},…………….

}

V = {a,b,c,d,e,f,g,h,i}

P(V) =

{

{a,b,c,d,e,f,g,h,i},

……………………..

{a,b,c,g},

{c,d,e,f},

…………..

{a,b,g},

{a,b,i},

{c,e,f},

{c,e,d},

{c,d,f},

{e,d,f},

……….

{a,b},{a,g},{b,g},{a,i},

{g,i},**{i,h},{g,h},{b,c},{c,d},**

{c,f},{e,f},{d,e},{c,e},{f,d},

{a},{b},{c},…………….

}

Clique List, **S**

S =

{

{c,d,e,f},

{a,b,g}, {c,e,f},

{c,e,d}, {c,d,f},

{e,d,f}, {a,b}, {a,g},

{b,g}, {a,i}, {i,h},

{g,h}, {b,c}, {c,d},

{c,f}, {e,f}, {d,e},

{c,e}, {f,d}, {a},

{b}, {c}, ………………..

}

S =

{

{c,d,e,f},

{a,b,g}, {c,e,f},

{c,e,d}, {c,d,f},

{e,d,f}, {a,b}, {a,g},

{b,g}, {a,i}, {i,h},

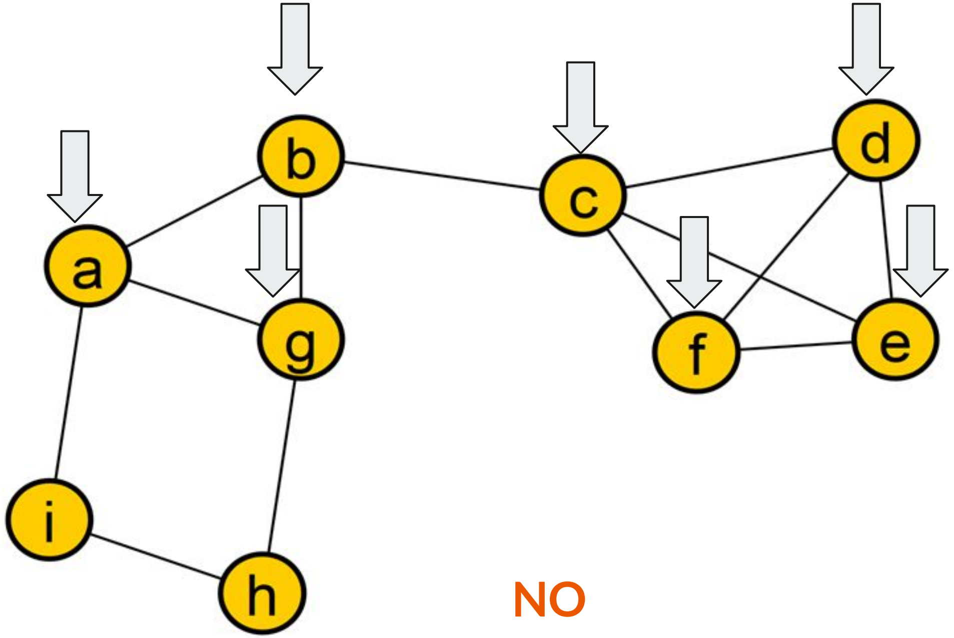
{g,h}, {b,c}, {c,d},

{c,f}, {e,f}, {d,e},

{c,e}, {f,d}, {a},

{b}, {c}, ………………..

}



NO

S =

{

{c,d,e,f},

{a,b,g}, {c,e,f},

{c,e,d}, {c,d,f},

{e,d,f}, {a,b}, {a,g},

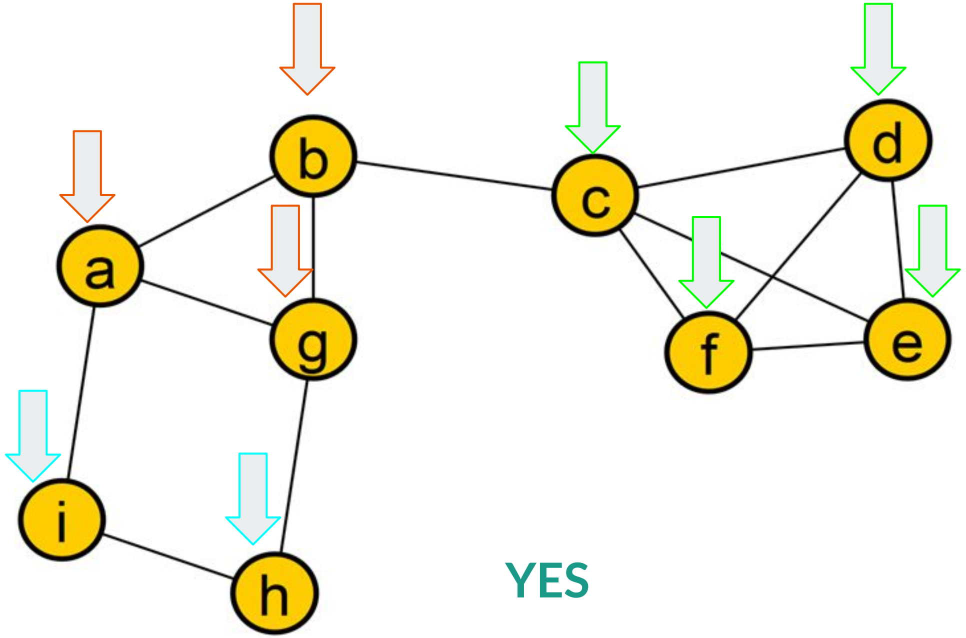
{b,g}, {a,i}, **{i,h}**,

{g,h}, {b,c}, {c,d},

{c,f}, {e,f}, {d,e},

{c,e}, {f,d},

}



YES

Time Complexity

Step 1 : Find the power set of **V** where **V** is the set of vertices of graph **G**.
Step 2 : Take an empty list **S**.
Step 3 : Take every element(which is actually a subset of V) of **P(V)** and check whether it's a complete subgraph or clique then add this to list **S** otherwise skip the element.
Step 4 : Find all possible subsets of **S** of size **k**.
Step 5 : Take every subsets and check whether it covers all vertices of **V** or not.

Step 1,2 and 3 runs in **$O(n^2 2^n)$**
Step 4 runs in **$O(2^{kn})$**
Step 5 runs in **$O(n2^{kn})$**

The time complexity of brute force algorithm is $O(n2^{kn})$

Exponential Time Exact Algorithm

- For the Edge Clique Cover problem there exist an exact exponential algorithm of complexity O*(2^n) Link.
- However for Vertex Clique Cover, “set partitioning via inclusion-exclusion” paper is more generalization of the problem of Vertex Clique Cover. So, we will use their idea to find an exact exponential algorithm of complexity O*(2^n) Link.
- The dual problem of vertex clique cover is K-Coloring Problem. When it is parameterized by number of colors, it is para-NP-hard. But when parameterized by vertex cover or treewidth it is in FPT. So same applies for Vertex Clique Cover. Link.

Prerequisite for O*(2^n) solution

- **Zeta Transform (SOS DP/Yate's DP):** Given an function F: $(0, 2^n]$ -> Integer. Compute zeta function of F in $O(n2^n)$, ie z(s) defined as,

  z(s) = Sum for all F(r) such that r is a subset of s.

- **Inclusion Exclusion:** Given n object each with or without property x where x = 1, 2 …. n. Then

  $n(A_1$ U …. A_n) = Sum of all $(-1)^{x+1}$ n($A_{i1} \cap \ldots \cap A_{ix}$) for 1 <= i1 …. <= ix <= n and x = 1 to n.

  This can be easily done in $O(2^n)$

Sum over Subset DP:

- We will represent every subset of a set of cardinality n with n bits. I’th bit is 1 of i’th element is in the subset, 0 otherwise. Example: S = {1,2,3,4}, A = {2, 3, 4}. Then A and S both are subset of S. Thus we represent S as 1111 and A as 1110. (We number the bit from right to left, 1 based)
- So z(1110) = f(1110) + f(1100) + f(1010) + f(0110) + f(1000) + f(0100) + f(0010) + f(0000)
- We will present $O(4^n)$, $O(3^n)$, and $O(n2^n)$ solution of SOS.

Brute Force Solution

```
for(int mask = 0; mask < (1<<n); ++mask)
      for(int i = 0;i < (1<<n); ++i)
            if((mask&i) == i)
                  Z[mask] += f[i];
```

This solution is quite straightforward and inefficient with time complexity of $O(4^n)$

Suboptimal Solution

```
for (int mask = 0; mask < (1<<n); mask++){
    Z[mask] = f[0];
    // iterate over all the subsets of the mask
    for(int i = mask; i > 0; i = (i-1) & mask){
         Z[mask] += f[i];
}
```

If a mask has K bit on, we iterate 2^k in inner loop. Thus giving us complexity, sum of ${}^nC_k*2^k$ for k = 0 to n. Which sums to $O(3^n)$ (easily done via binomial)

Optimal Solution

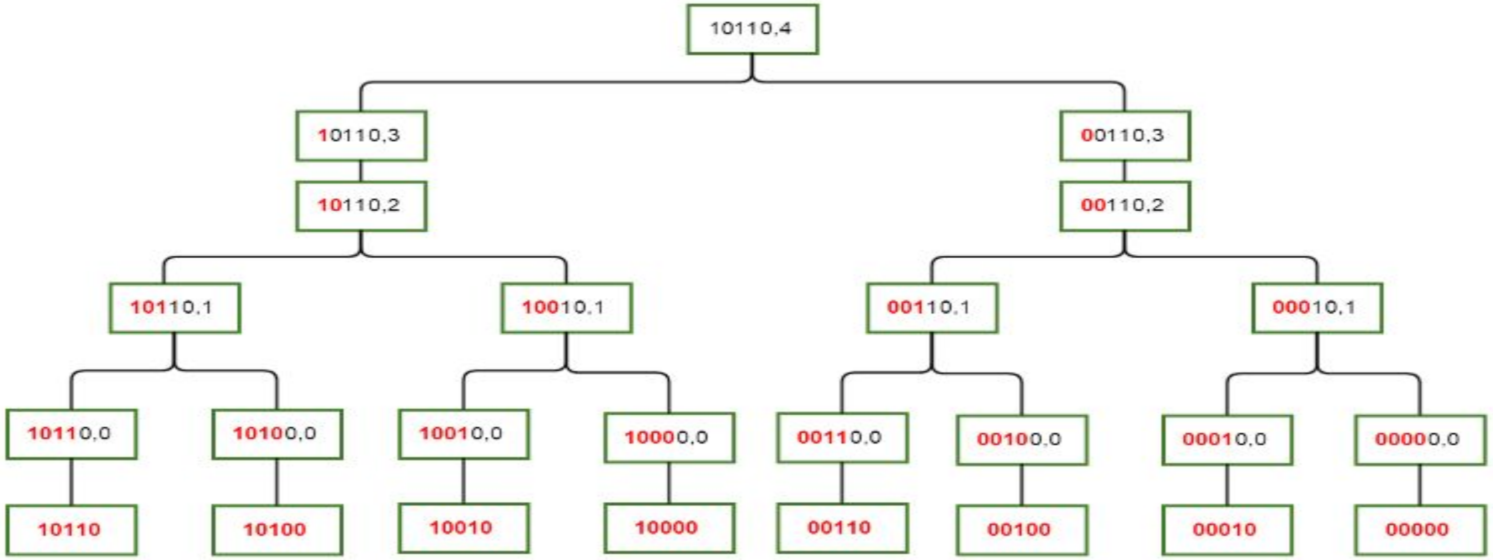
- S(mask, i) = that is set of only those subsets of **mask** which *differ* from **mask** only in the first **i** bits (zero based). For example S(101**0101**, 3) = { 101**0101**, 101**0100**, 101**0010**, 101**0000** }

$$S(mask,\ i) = \begin{cases} S(mask,\ i-1) & i^{th}\ bit\ OFF \\ S(mask,\ i-1) \bigcup S(mask \oplus 2^i,\ i-1) & i^{th}\ bit\ ON \end{cases}$$

Example: S(101**0**101, 3) = S(1010101, 2), As the 3rd bit (0 based) is zero

S(101**1**101, 3) = S(101**1**101, 2) + S(101**0**101, 2), As the 3rd bit (0 based) is one

Optimal Solution



Optimal Solution

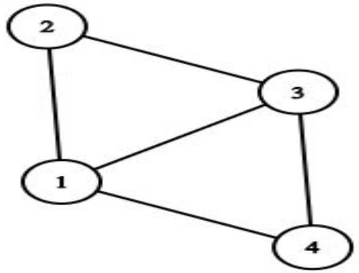
```
for(int i = 0; i<(1<<n); ++i)
      Z[i] = f[i];
for(int i = 0;i < n; ++i)
      for(int mask = 0; mask < (1<<N); ++mask){
            if(mask & (1<<i))
                  Z[mask] += Z[mask^(1<<i)];

}
```

Memory Complexity: **$O(2^n)$**

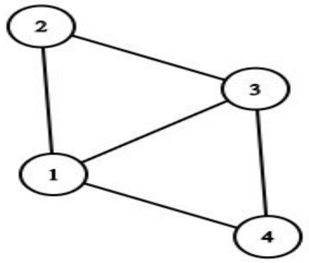
Time Complexity: **$O(n2^n)$**

Solution For Clique Cover

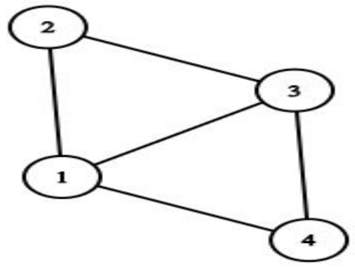


- First we will list all the cliques of the target graph and make F(clique) = 1. This can be done in **$O(n^2 2^n)$**
- As example for the above graph we have, F(0001) = 1, F(0010) = 1, F(0100) = 1, F(1000) = 1, F(0011) = 1, F(0110) = 1, F(0101) = 1, F(1001) = 1, F(1100) = 1, F(0111) = 1, F(1101) = 1. Here F(1101) corresponds to subgraph induced by {1,3,4}, Rest of the F(mask) = 0.
- Now Calculate the zeta values (ie sos dp) on F.
- As example for the above graph z(0101) = F(0101) + F(0100) + F(0001) + F(0000)
- we have, z(0000) = 0, z(0001) = 1, z(0010) = 1, z(0100) = 1, z(1000) = 1, z(0011) = 3, z(0101) = 3, z(1001) = 3, z(0110) = 3, z(1010) = 2, z(1100) = 3, z(0111) = 7, z(1011) = 5, z(1101) = 7, z(1110) = 5. What the z values indicates? 0 means not using that node, 1 means using that node. So z(0111) indicates number of subgraph of {1,2,3} which are cliques.

Solution For Clique Cover



- Now we solve the following question. Given a graph G(V,E) and a number k, we need to answer yes or no if we can partition V(G) in K cliques.
- Define object as a set of k cliques (clique can repeat). As example for above graph, { {1,2}, {2,3} }, { {1,3,4}, {1} } are two set for k = 2.
- So if a graph has x cliques then, it has x^k such sets (or objects)
- Now, define A_i = number of objects (or sets) where no clique has vertex i.
- So n(A_1 U A_2 U ….. U A_n) indicates number of such sets where all cliques in any set lack at least one of the vertex 1,2 .. . n.
- So x^k - n(A_1 U A_2 U ….. U A_n) indicates the number of such set where all clique in that set cover V(G).



Solution For Clique Cover

- For example, $n(A_1) = z(1110)^k$, $n(A_2 \cap A_4) = z(0101)^k$. Why is that? z(0101) indicates the number of clique where none has vertex 2, 4. So $z(0101)^k$ indicates number of k size sets where no set has vertex 2, 4 which is $n(A_2 \cap A_4)$.
- Now we can easily calculate $n(A_1 \cup A_2 \cup \ldots.. \cup A_n)$ using z values. Thus $x^k - n(A_1 \cup A_2 \cup \ldots.. \cup A_n)$. If the value of $x^k - n(A_1 \cup A_2 \cup \ldots.. \cup A_n)$ is greater than zero we will output yes, no otherwise.
- For the above graph with k = 2 we have x = 11. So $x^k = 121$. Now $n(A_1 \cup A_2 \cup A_3 \cup A_4) = 5^2 + 7^2 + 5^2 + 7^2 - 3^2 - 3^2 - 3^2 - 3^2 - 3^2 - 2^2 + 1^2 + 1^2 + 1^2 + 1^2 = 103$. Thus 121 - 103 = 18. So we will output yes for k = 2 in above graph.
- Finding all clique takes $O(n^2 2^n)$, Zeta value calculation takes $O(n2^n)$ and inclusion-exclusion takes $O(2^n)$. Thus giving total complexity $\mathbf{O(n^2 2^n)}$ or $\mathbf{O^*(2^n)}$

Algorithms for restricted classes

- Perfect graphs are defined as the graphs in which, for every induced subgraph, the chromatic number (minimum number of colors in a coloring) equals the size of the maximum clique.
- According to the weak perfect graph theorem, the complement of a perfect graph is also perfect. Therefore, the perfect graphs are also the graphs in which, for every induced subgraph, the clique cover number equals the size of the maximum independent set.
- It is possible to compute the clique cover number in perfect graphs in polynomial time. Link
- Also minimum clique cover for triangle free graph can also be solved in polynomial time.

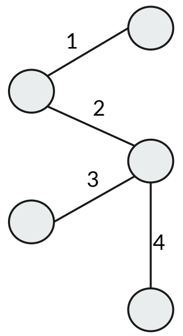
Algorithm for triangle free graph

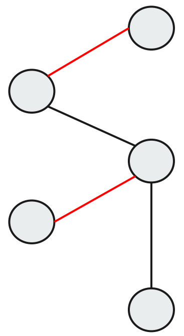
Input: A triangle free graph **G**(**V**, **E**)
Output: Sets of vertices where each set is a clique of graph **G**

```
1. R ← { }
2. T ← { }
3. M ← MaximumMatching(G)   } O(√VE)
4. for each (u, v) ϵ M
5.       R ← R U { {u, v} }   }
6.       T ← T U {u, v}       } O(V)
7. R ← R U (V - T)   } O(V)
8. return R
```

Complexity: O(√VE + V)

Example...





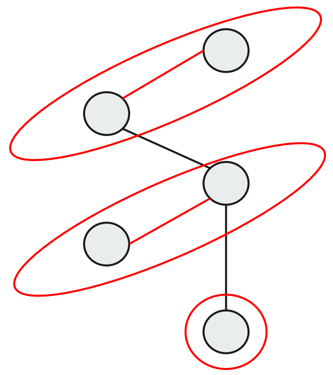


Fig: Triangle free graph

Another Example...

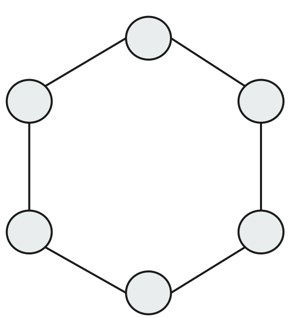
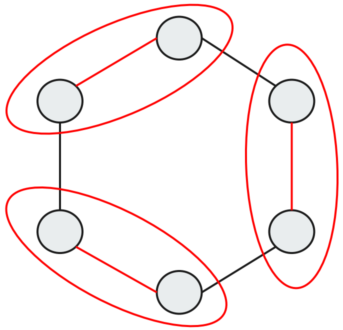



Fig: C_6

That's All

Any Questions?