Exercises

→ consider a search tree with branching factor = 4 and height $n/3$. What are #solns considered by the tree?

$$= 4^{n/3} \approx (1.589)^{n}$$

→ vertex cover using naive search tree

```
def rec_searchtree (graph, assign)

    n = len (graph)

    for i in range (0, n):
        for j in range (i, n):
            if graph[i][j] == 1:
                if assign[i] == 0 and assign[j] == 0:
                    return float ("inf")

    if None not in assign:
        size = sum (assign)
        return size

    v = assign.index (None)

    assign[v] = 0
    size_v_0 = rec_search_tree (graph, assign)
    assign[v] = 1
    size_v_1 = rec_search_tree (graph, assign)
    asssign[v] = None
    return min (size_v_0, size_v_1)
```

→ previous search tree is $O(2^n)$. Let's improve that

Instead of finding one vertex, let's find 2 vertices.

```
def rec_stree(graph, assign):

    n = len(graph)
    u = -1
    v = -1
    for i in range(0, n):

        for j in range(i+1, n):

            if graph[i][j] == 1:

                if assignment[i] == 0 and assignment[j]
                    return float("inf")

                elif assign[i] == None and assign[j] == No
                    u = i
                    v = j

    # 3 cases
    # u and v are ≥ 0
    # one of u and v are ≥ 0, we can figure the assignme
    # both are -1, return sum(assign)

    if u == -1 and v == -1:
        return sum(assign)

    if u != -1 and v != -1:
        assign[u] = 1
        assign[v] = 0
        size_10 = rec_stree(graph, assign)
        assign[u] = 0
        assign[v] = 1
        size_01 = rec_stree(graph, assign)

        assign[u] = 1
        assign[v] = 1
        size_11 = rec_stree(graph, assign)
        assign[u] = None
        assign[v] = None
        return min(size_10, size_01, size_11)
```

```
else:  # case
    # if  u == -1:
        v = u
    assign[v] = 0
    for  i  in range [0, n):
        if  graph[v][:] == 1 and assign[i] == 0:
            assign[v] = 1
            return size
    return sum(assign)
```

Fixed parameter tractable problems

parameter k

Graph G

#vertices n

V.C.

At most k vertices are needed to cover all edges

If any vertex is an end point to atleast $k+1$ edges it must be in the cover (otherwise, we'd need more than k vertices in the vertex cover)

All remaining vertices can have at most k edges if put in the cover

∴ The remaining graph has k^2 edges at the most for G to be an YES instance of V.C.

(k vertices, each has at most k edges)