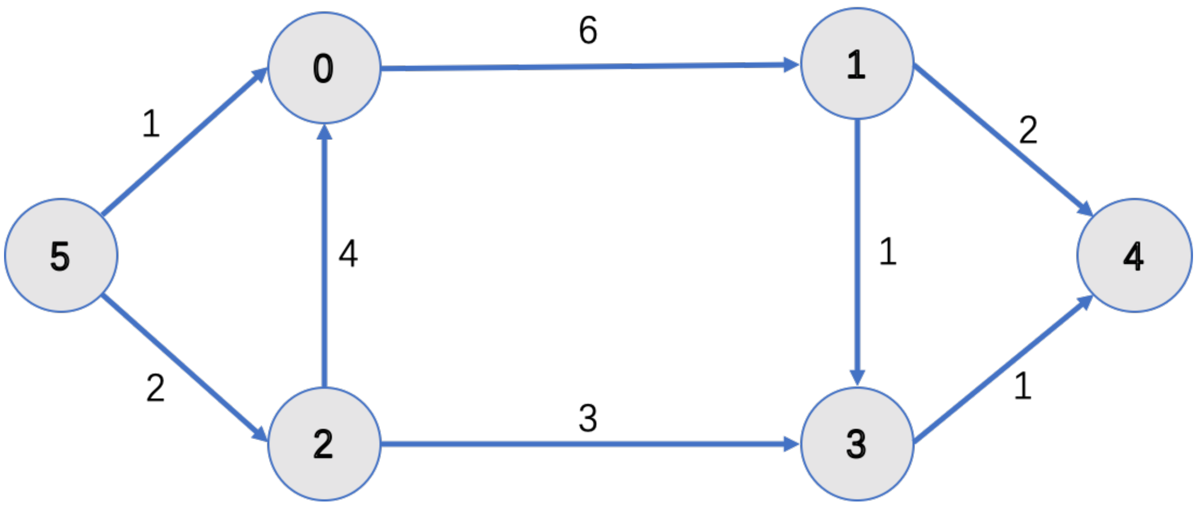
Yuan Gao z5239220 Q4

Given a weighted directed graph $G(V, E)$, find a path in G (possibly self-intersecting) of length exactly K that has the maximum total weight.

1. Abstract graph

> The data structure of the graph is:
>
> ```
> class Graph{
> private:
>   int verTexCount;  //number of vertices
>   int edgeCount;  //the number of edges
>   vector<int> verTexs; //vertices array
>   vector<vector<int>> edge; //adjacency Matrix
> };
> ```

> For example: the graph
>
> 
>
>
> Can be abstracted to:
>
> ```
> edge = {
>     {0, 6, 0, 0, 0, 0},
>     {0, 0, 0, 1, 2, 0},
>     {4, 0, 0, 3, 0, 0},
>     {0, 0, 0, 0, 1, 0},
>     {0, 0, 0, 0, 0, 0},
>     {1, 0, 2, 0, 0, 0}
> };
> ```

2. Dynamic programming

> - Initialize dynamic programming array

> Let `dp[start][end][K]` be the longest path from `start` vertex to the `end` vertex going through exactly `K` edges in total.
> Therefore, in `dp[i][j][k]`, $i, j \in [0, verTexCount]$ and $k \in [0, K]$.
>
> > `k = 0`: `dp[i][i][0] = 0`
> > `k = 1`: `dp[i][j][1] = edge[i][j]`

- Recursion

> The main dynamic programming function is:
>
> > `dp[start][i][k + 1] = max(dp[start][i][k + 1], dp[start][curr][k] + edge[curr][i]);`
> > Use BFS algorithm to control current vertex and its adjacent vertices
> > Make sure every start vertex has K iterations

- Result
  The maximum value in `dp[i][j][k + 1]` sequence

3. Code

```
/*Create graph data structure*/
class Graph
{
private:
    int verTexCount;  //number of vertices
    int edgeCount;  //the number of edges
    vector<int> verTexs; //vertices array
    vector<vector<int>> edge; //adjacency Matrix

public:
    /*Help fuction(not show)*/
    Graph(int n,int m);
    ~Graph();
    void insert_vertex(int num, int v);
    void insert_edge(int u, int v);
    void show_Graph();

    /*Mian function*/
    bool GraphIsAdjacent(int u, int v);
    void BFS(vector<vector<vector<int>>> &dp, int start, int K);
    int maxTotalWeight(int K);
};

/*Judge whether vertex u, v are adjacent vertex*/
bool Graph::GraphIsAdjacent(int u, int v){
    return edge[u][v];
}

/*BFS algorithm to control current vertex and its adjacent vertices(from
'start' vertex move 'K' times)*/
```

```
void Graph::BFS(vector<vector<vector<int>>> &dp, int start, int K){
    queue<int> element;
    int res = 0;
    int k = 1;
    element.push(start);
    while(!element.empty()){
        //Just loop exactly K times
        if(k > K){
            return;
        }

        //Make sure traverse every adjacent vertex of current vertex
        int level = element.size();
        for(int n = 0; n < level; ++n){
            int curr = element.front();
            element.pop();

            //Search maximum weight path between 'start' to 'i'
            for(int i = 0; i < verTexCount; ++i){
                if(GraphIsAdjacent(curr, i)){
                    if(dp[start][curr][k] + edge[curr][i] > dp[start][i][k +
1]){
                        dp[start][i][k + 1] = dp[start][curr][k] + edge[curr]
[i];
                        element.push(i);
                    }
                }
            }
            //if current level finished, step to next level
            if(n == level - 1){
                k++;
            }
        }
    }
}

int Graph::maxTotalWeight(int K){
    //init dp[] array
    vector<vector<vector<int>>> dp(verTexCount, vector<vector<int>>
(verTexCount, vector<int>(verTexCount, 0)));
    for(int i = 0; i < verTexCount; ++i){
        for(int j = 0; j < verTexCount; ++j){
            dp[i][j][1] = edge[i][j];
        }
    }

    //start from every vertex
    int maxRes = 0;
    for(int i = 0; i < verTexCount; ++i){
```

```
            BFS(dp, verTexs[i], K);
        }

        //get the total weight
        for(int i = 0; i < verTexCount; ++i){
            for(int j = 0; j < verTexCount; ++j){
                if(dp[i][j][K + 1] > maxRes){
                    maxRes = dp[i][j][K + 1];
                }
            }
        }
        return maxRes;
    }
```

4. Time complexity

> Assume the graph has n vertices
>
> > -BFS algorithm costs $O(K \times n)$
> > -Total graph traversal costs $O(n \times (K \times n))$
> > -Get maximum total weight costs $O(n^2)$

> Hence, total time complexity is $O(K \times n^2)$