The FarshadBazi Code Notebook

December 18, 2019

Contents

0.1 Graph

0.1.1 BFS

```cpp
#include <bits/stdc++.h>
using namespace std;
const int SIZE = 100; //Vertex size
int main(){
  int source = 0;
  vector<int> Graph[SIZE];
  bool visited[SIZE];
  int distance[SIZE];
  fill(visited, visited+SIZE, 0);
  fill(distance, distance+SIZE, INT_MAX);
  queue<int> q;
  q.push(source);
  visited[source] = true;
  distance[source] = 0;
  while(!q.empty()){
    int u = q.front();
    q.pop();
    for(int i = 0 ; i < Graph[u].size(); i++){
      if(visited[Graph[u][i]]){
        continue;
      }
      visited[Graph[u][i]] = true;
      distance[Graph[u][i]] = distance[u] + 1;
      q.push(Graph[u][i]);
    }
  }
  return 0;
}
```

0.1.2 Topological-Sorting

```cpp
#include <bits/stdc++.h>
using namespace std;
const int SIZE = 100;//Graph Size Vertex
int visited[SIZE];
vector <int> Graph[SIZE];
stack<int> ans;
int topolSort(int u)
{
  if(visited[u] == 1)
  {
    return -1;//cycle
  }
  if(visited[u] == 2)
  {
    return 0;
  }
  visited[u] = 1;
  for(int i = 0 ; i < Graph[u].size() ; i++)
  {
    topolSort(Graph[u][i]);
  }
    visited[u] = 2;
  ans.push(u);
  return 0;
}

int main(){
  return 0;
}
```

0.1.3 DAG Shortest Path

```cpp
#include<iostream>
#include <vector>
using namespace std;

const int Gsize=100000;

int _color[Gsize], _p[Gsize] ,_d[Gsize] ,_f[Gsize] , _time;
vector<int> _V;
void DFS_visit(vector<int> G[] ,int u)
{
  _time++;
  _d[u]=_time;
  _color[u]=1;
  int v;
  for(int i=0;i<G[u].size();i++)
  {
    v=G[u][i];
    if(!_color[v])
    {
      _p[v]=u;
      DFS_visit(G ,v);
    }
  }
  _color[u]=2;
  _time++;
  _f[u]=_time;
  _V.push_back(u);
}
void DFS(vector<int>G[])
{
  for(int i=0;i<Gsize;i++)
  {
    _color[i]=0; _p[i]=0;
  }
  _time=0;
  for(int i=0;i<Gsize;i++)
    if(!_color[i])
      DFS_visit(G ,i);
}

void Topologival_sort(vector<int> G[], int arr[])
{
  _V.clear();
  DFS(G);
  for(int i=0;i<_V.size();i++)
    arr[_V.size()-1-i] = _V[i];
}

void _Relax(int u, int v, int w)
{
  if(_d[v]>(_d[u]+w))
  {
    _d[v] = _d[u]+w;
    _p[v] = u;
  }
}

void DAG_shortest_paths(vector<int> G[], vector<int> W[], int s)
{
  int arr[Gsize];
  Topologival_sort(G, arr);
  for(int i=0;i<Gsize;i++)
    _d[i]=500000000;
```

```
  _d[s] = 0;
  for(int i=0;i<Gsize;i++)
  {
    int u=arr[i];
    for(int j=0;j<G[u].size();j++)
      _Relax(u, G[u][j], W[u][j]);
  }
}

int main()
{

  return 0;
}
```

0.1.4 Bellman-Ford

```
// A C++ program for Bellman-Ford's single source
#include <bits/stdc++.h>

struct Edge
{
    int src, dest, weight;
};

struct Graph
{
    int V, E; //Size Of Graph
    struct Edge* edge;
};

struct Graph* createGraph(int V, int E)
{
    struct Graph* graph = new Graph;
    graph->V = V;
    graph->E = E;
    graph->edge = new Edge[E];
    return graph;
}

void BellmanFord(struct Graph* graph, int src, int* dist)
{
    int V = graph->V;
    int E = graph->E;

    for (int i = 0; i < V; i++)
        dist[i]   = INT_MAX;
    dist[src] = 0;

    for (int i = 1; i <= V-1; i++)
    {
        for (int j = 0; j < E; j++)
        {
            int u = graph->edge[j].src;
            int v = graph->edge[j].dest;
            int weight = graph->edge[j].weight;
            if (dist[u] != INT_MAX && dist[u] + weight < dist[v])
                dist[v] = dist[u] + weight;
        }
    }

    for (int i = 0; i < E; i++)
    {
        int u = graph->edge[i].src;
        int v = graph->edge[i].dest;
```

```
        int weight = graph->edge[i].weight;
        if (dist[u] != INT_MAX && dist[u] + weight < dist[v])
            printf("Graph contains negative weight cycle");//Hint!
    }

    //printArr(dist, V);//Finishing Algo

    return;
}

int main()
{
    int V = 5;  // Number of vertices in graph
    int E = 8;  // Number of edges in graph
    struct Graph* graph = createGraph(V, E);
    graph->edge[0].src = 0;
    graph->edge[0].dest = 1;
    graph->edge[0].weight = -1;
    int dist[5];
    BellmanFord(graph, 0, dist);
    return 0;
}
```

0.1.5 Djkstra

```
#include <bits/stdc++.h>
using namespace std;
#define endl '\n'
#define pii pair<int,int>
#define F first
#define S second
#define mp make_pair
#define pb emplace_back

bool vis[100001];
int dis[100001];
vector<pii> a[100001];

class prioritize {
public: bool operator ()(pii &p1 , pii &p2) {
    return p1.S > p2.S;
  }
};

int Dijkstra(int s, int n) {
  for (int i = 0; i <= n; i++) {
    vis[i] = false;
    dis[i] = INT_MAX;
  }
  priority_queue<pii, vector<pii>, prioritize> pq;
  pq.push(mp(s, dis[s] = 0));
  while (!pq.empty()) {
    pii cur = pq.top(); pq.pop();
    int cv = cur.F, cw = cur.S;
    if (vis[cv]) continue;
    vis[cv] = true;
    for (pii x : a[cv]) {
      if (!vis[x.F] && (cw + x.S) < dis[x.F]) {
        pq.push(mp(x.F, dis[x.F] = cw + x.S));
      }
    }
  }
}

int main() {
```

```
  int tc;
  cin >> tc;
  while (tc--) {
    int v1, v2, w, n, m;
    cin >> n >> m;
    for (int i = 0; i <= n; i++) {
      a[i].clear();
    }
    for (int i = 0; i < m; i++) {
      cin >> v1 >> v2 >> w;
      a[v1].pb(mp(v2, w));
    }
    int s;
    cin >> s;
    Dijkstra(s, n);
    for (int i = 1; i <= n; i++) {
      if (dis[i] != INT_MAX) {
        cout << dis[i] << "␣";
      } else {
        cout << "-1␣";
      }
    }
  }
  return 0;
}
```

0.1.6 Floyd Warshall

```
// C Program for Floyd Warshall Algorithm
#include<stdio.h>

#define V 4
#define INF 9999999

void floydWarshall (int graph[][V])
{
  int dist[V][V], i, j, k;
  for (i = 0; i < V; i++)
    for (j = 0; j < V; j++)
      dist[i][j] = graph[i][j];
    for (k = 0; k < V; k++)
    {
      for (i = 0; i < V; i++)
      {
        for (j = 0; j < V; j++)
        {
          if (dist[i][k] + dist[k][j] < dist[i][j])
              dist[i][j] = dist[i][k] + dist[k][j];
        }
      }
    }
    // All distances -> dist
}

int main()
{
    /* example Graph:
             10
       (0)-------->(3)
        |          /|\
      5 |           |
        |           | 1
       \|/          |
       (1)-------->(2)
             3           */
```

```
    int graph[V][V] =
    {
      {0,   5,  INF, 10},
      {INF, 0,   3, INF},
      {INF, INF, 0,   1},
      {INF, INF, INF, 0}
    };

    // Print the solution
    floydWarshall(graph);
    return 0;
}
```

0.1.7 Strongly Connected Component

```
#include <bits/stdc++.h>
using namespace std;
const int SIZE = 50001;
bool visited[SIZE];
//Input: vector<int> Graph, vector<int> Reverce Graph,
// int Nomber Of vertexces
//Output:
//Strongly connected componnet Graph->
//--->SCC_Graph
//List of Each Componnent -->
//----->ListOfEachSCC
//to see each node blongs to where
//------SCC_list
//SCC returns the size of SCC graph
vector<int> SCC_Graph[SIZE];
vector<int> ListOfEachSCC[SIZE];
int SCC_List[SIZE];

void TopolSort(vector<int> g[],int u, stack<int>& ans){
    if(visited[u]){
        return;
    }
    visited[u] = true;
    for(int i = 0 ; i < g[u].size() ; i++){
        TopolSort(g,g[u][i],ans);
    }
    ans.push(u);
}

void DFSR(vector<int> g[],int u, int counter){
    if(visited[u]){
        return;
    }
    visited[u] = true;
    for(int i = 0 ; i < g[u].size() ; i++){
        DFSR(g,g[u][i], counter);
    }
    ListOfEachSCC[counter].push_back(u);
    SCC_List[u] = counter;
}

int SCC(vector<int> graph[], vector<int> graph_reverce[],int v){
    fill(visited, visited+v, false);
    for(int i = 0 ; i < v ; i++){
        SCC_Graph[i].clear();
        ListOfEachSCC[i].clear();
        SCC_List[i] = 0;
    }
    stack<int> TopolSorted;
    for(int i = 0 ; i < v ; i++){
```

```
        if(visited[i]){
            continue;
        }
        TopolSort(graph,i, TopolSorted);
    }
    fill(visited, visited+v, false);
    fill(SCC_List, SCC_List+v, 0);
    int counter = 0;
    while(!TopolSorted.empty()){
        int u = TopolSorted.top();
        TopolSorted.pop();
        if(visited[u]){
            continue;
        }
        DFSR(graph_reverce, u, counter);
        counter++;
    }
    for(int i = 0 ; i < counter ; i++){
        for(int j = 0 ; j < ListOfEachSCC[i].size() ; j++){
            int u = ListOfEachSCC[i][j];
            for(int k = 0 ; k < graph[u].size() ; k++){
                int w = graph[u][k];
                if(SCC_List[u] != SCC_List[w]){
                    SCC_Graph[SCC_List[u]].push_back(SCC_List[w]);
                }
            }
        }
    }
    return counter;
}
```

0.1.8 Minimum Spanning Tree

```
//C++ program for Prim's Minimum Spanning Tree (MST) algorithm.
#include <stdio.h>
#include <limits.h>

#define V 5 //Graph Size

int minKey(int key[], bool mstSet[])
{
   int min = INT_MAX, min_index;
   for (int v = 0; v < V; v++)
     if (mstSet[v] == false && key[v] < min)
         min = key[v], min_index = v;
   return min_index;
}

int printMST(int parent[], int n, int graph[V][V])
{
   printf("Edge   Weight\n");
   for (int i = 1; i < V; i++)
      printf("%d - %d     %d \n", parent[i], i, graph[i][parent[i]]);
}

void primMST(int graph[V][V])
{
    int parent[V];
    int key[V];
    bool mstSet[V];
    for (int i = 0; i < V; i++)
        key[i] = INT_MAX, mstSet[i] = false;
    key[0] = 0;
    parent[0] = -1;
    for (int count = 0; count < V-1; count++)
```

```
    {
        int u = minKey(key, mstSet);
        mstSet[u] = true;
        for (int v = 0; v < V; v++)
          if (graph[u][v] && mstSet[v] == false && graph[u][v] <  key[v])
             parent[v]  = u, key[v] = graph[u][v];
    }
    printMST(parent, V, graph);//print Solution
}

int main()
{
   /* Let us create the following graph
            2    3
       (0)--(1)--(2)
        |   / \   |
       6| 8/   \5 |7
        | /     \ |
       (3)-------(4)
             9          */
    int graph[V][V] =
     {{0, 2, 0, 6, 0},
     {2, 0, 3, 8, 5},
     {0, 3, 0, 0, 7},
     {6, 8, 0, 0, 9},
     {0, 5, 7, 9, 0},
     };
     primMST(graph);
     return 0;
}
```

0.1.9 MST Kruskal

```
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;

class Disjoint_set
{
  int *id, cnt, *sz;
public:
  Disjoint_set(int N)
  {
    cnt = N;
    id = new int[N];
    sz = new int[N];
    for(int i=0; i<N; i++)
    {
      id[i] = i;
      sz[i] = 1;
    }
  }
  ~Disjoint_set()
  {
    delete [] id;
    delete [] sz;
  }
  int find(int p)
  {
    int root = p;
    while (root != id[root])
      root = id[root];
    while (p != root)
    {
```

```
33        int newp = id[p];
34        id[p] = root;
35        p = newp;
36      }
37      return root;
38    }
39    void merge(int x, int y)
40    {
41      int i = find(x);
42      int j = find(y);
43      if (i == j) return;
44      if   (sz[i] < sz[j])
45      {
46        id[i] = j;
47        sz[j] += sz[i];
48      }
49      else
50      {
51        id[j] = i;
52        sz[i] += sz[j];
53      }
54      cnt--;
55    }
56    bool connected(int x, int y) {return find(x) == find(y);}
57    int count() {return cnt;}
58  };
59
60  const int Gsize = 9;
61
62  bool cmp(pair<pair<int, int>, int> p1, pair<pair<int, int>, int> p2)
63  {
64    return p1.second<p2.second;
65  }
66
67  vector<pair<int, int> > _A;
68
69  void MST_kruskal(vector<int> G[], vector<int> w[])
70  {
71    Disjoint_set set(Gsize);
72    _A.clear();
73    vector<pair<pair<int, int>, int> > E;
74    for(int i=0;i<Gsize;i++)
75      for(int j=0;j<G[i].size();j++)
76        if(i<G[i][j])
77          E.push_back(make_pair(make_pair(i, G[i][j]), w[i][j]));
78    sort(E.begin(), E.end(), cmp);
79    for(int i=0;i<E.size();i++)
80      if(!set.connected(E[i].first.first, E[i].first.second))
81      {
82        _A.push_back(make_pair(E[i].first.first, E[i].first.second));
83        set.merge(E[i].first.first, E[i].first.second);
84      }
85  }
86
87  void MST_kruskal(vector<pair<pair<int, int>, int> > E)
88  {
89    Disjoint_set set(Gsize);
90    _A.clear();
91    sort(E.begin(), E.end(), cmp);
92    for(int i=0;i<E.size();i++)
93      if(!set.connected(E[i].first.first, E[i].first.second))
94      {
95        _A.push_back(make_pair(E[i].first.first, E[i].first.second));
96        set.merge(E[i].first.first, E[i].first.second);
97      }
```

```
}

int main()
{

  return 0;
}
```

0.1.10 Maximum-BPM

```
// A C++ program to find maximal Bipartite matching.
#include <iostream>
#include <string.h>
using namespace std;

#define M 6 // Size Of Graph M*N
#define N 6

bool bpm(bool bpGraph[M][N], int u, bool seen[], int matchR[])
{
    for (int v = 0; v < N; v++)
    {
        if (bpGraph[u][v] && !seen[v])
        {
            seen[v] = true; // Mark v as visited
            if (matchR[v] < 0 || bpm(bpGraph, matchR[v], seen, matchR))
            {
                matchR[v] = u;
                return true;
            }
        }
    }
    return false;
}

int maxBPM(bool bpGraph[M][N])
{
    int matchR[N];//List Of Matches

    memset(matchR, -1, sizeof(matchR));

    int result = 0;
    for (int u = 0; u < M; u++)
    {
        bool seen[N];
        memset(seen, 0, sizeof(seen));
        if (bpm(bpGraph, u, seen, matchR))
            result++;
    }
    return result;
}

int main()
{
    bool bpGraph[M][N] = {  {0, 1, 1, 0, 0, 0},
                            {1, 0, 0, 1, 0, 0},
                            {0, 0, 1, 0, 0, 0},
                            {0, 0, 1, 1, 0, 0},
                            {0, 0, 0, 0, 0, 0},
                            {0, 0, 0, 0, 0, 1}
                         };

cout << "Maximum␣Maching␣"<< maxBPM(bpGraph);

return 0;
```

```
}
```

0.2 Flow

0.2.1 Max-Flow

```
// C++ program for implementation of Ford Fulkerson algorithm
#include <iostream>
#include <limits.h>
#include <string.h>
#include <queue>
using namespace std;

#define V 6

bool bfs(int rGraph[V][V], int s, int t, int parent[])
{
    bool visited[V];
    memset(visited, 0, sizeof(visited));
    queue <int> q;
    q.push(s);
    visited[s] = true;
    parent[s] = -1;
    while (!q.empty())
    {
        int u = q.front();
        q.pop();

        for (int v=0; v<V; v++)
        {
            if (visited[v]==false && rGraph[u][v] > 0)
            {
                q.push(v);
                parent[v] = u;
                visited[v] = true;
            }
        }
    }
    return (visited[t] == true);
}

int fordFulkerson(int graph[V][V], int s, int t)
{
    int u, v;
    int rGraph[V][V];
    for (u = 0; u < V; u++)
        for (v = 0; v < V; v++)
             rGraph[u][v] = graph[u][v];
    int parent[V];
    int max_flow = 0;
    while (bfs(rGraph, s, t, parent))
    {
        int path_flow = INT_MAX;
        for (v=t; v!=s; v=parent[v])
        {
            u = parent[v];
            path_flow = min(path_flow, rGraph[u][v]);
        }

        for (v=t; v != s; v=parent[v])
        {
            u = parent[v];
            rGraph[u][v] -= path_flow;
            rGraph[v][u] += path_flow;
        }
```

```
        max_flow += path_flow;
    }
    return max_flow;
}

int main()
{
    int graph[V][V] = { {0, 16, 13, 0, 0, 0},
                        {0, 0, 10, 12, 0, 0},
                        {0, 4, 0, 0, 14, 0},
                        {0, 0, 9, 0, 0, 20},
                        {0, 0, 0, 7, 0, 4},
                        {0, 0, 0, 0, 0, 0}
                      };
    cout << "The maximum possible flow is "
     << fordFulkerson(graph, 0, 5);
    return 0;
}
```

0.2.2 Min-Cut

```
// C++ program for finding minimum cut using Ford-Fulkerson
#include <iostream>
#include <limits.h>
#include <string.h>
#include <queue>
using namespace std;

// Number of vertices in given graph
#define V 6

int bfs(int rGraph[V][V], int s, int t, int parent[])
{
    bool visited[V];
    memset(visited, 0, sizeof(visited));

    queue <int> q;
    q.push(s);
    visited[s] = true;
    parent[s] = -1;

    while (!q.empty())
    {
        int u = q.front();
        q.pop();

        for (int v=0; v<V; v++)
        {
            if (visited[v]==false && rGraph[u][v] > 0)
            {
                q.push(v);
                parent[v] = u;
                visited[v] = true;
            }
        }
    }
    return (visited[t] == true);
}

void dfs(int rGraph[V][V], int s, bool visited[])
{
    visited[s] = true;
    for (int i = 0; i < V; i++)
       if (rGraph[s][i] && !visited[i])
           dfs(rGraph, i, visited);
```

```
}

void minCut(int graph[V][V], int s, int t)
{
    int u, v;
    int rGraph[V][V];
    for (u = 0; u < V; u++)
        for (v = 0; v < V; v++)
             rGraph[u][v] = graph[u][v];
    int parent[V];
    while (bfs(rGraph, s, t, parent))
    {
        int path_flow = INT_MAX;
        for (v=t; v!=s; v=parent[v])
        {
            u = parent[v];
            path_flow = min(path_flow, rGraph[u][v]);
        }
        for (v=t; v != s; v=parent[v])
        {
            u = parent[v];
            rGraph[u][v] -= path_flow;
            rGraph[v][u] += path_flow;
        }
    }
    //Finishing...
    bool visited[V];
    memset(visited, false, sizeof(visited));
    dfs(rGraph, s, visited);
    for (int i = 0; i < V; i++)
      for (int j = 0; j < V; j++)
         if (visited[i] && !visited[j] && graph[i][j])
              cout << i << " - " << j << endl;

    return;
}

int main()
{
    int graph[V][V] = { {0, 16, 13, 0, 0, 0},
                        {0, 0, 10, 12, 0, 0},
                        {0, 4, 0, 0, 14, 0},
                        {0, 0, 9, 0, 0, 20},
                        {0, 0, 0, 7, 0, 4},
                        {0, 0, 0, 0, 0, 0}
                      };

    minCut(graph, 0, 5);

    return 0;
}
```

0.3 Geometry

0.3.1 Convex-Hull

```
// A C++ program to find convex hull of a set of points. Refer
#include <iostream>
#include <stack>
#include <stdlib.h>
using namespace std;

struct Point
{
    int x, y;
```

```
10  };
11
12  Point p0;
13
14  Point nextToTop(stack<Point> &S)
15  {
16      Point p = S.top();
17      S.pop();
18      Point res = S.top();
19      S.push(p);
20      return res;
21  }
22
23  int swap(Point &p1, Point &p2)
24  {
25      Point temp = p1;
26      p1 = p2;
27      p2 = temp;
28  }
29
30  int distSq(Point p1, Point p2)
31  {
32      return (p1.x - p2.x)*(p1.x - p2.x) +
33             (p1.y - p2.y)*(p1.y - p2.y);
34  }
35
36  int orientation(Point p, Point q, Point r)
37  {
38      int val = (q.y - p.y) * (r.x - q.x)
39              - (q.x - p.x) * (r.y - q.y);
40      if (val == 0) return 0;
41      return (val > 0)? 1: 2;
42  }
43
44  int compare(const void *vp1, const void *vp2)
45  {
46     Point *p1 = (Point *)vp1;
47     Point *p2 = (Point *)vp2;
48
49     int o = orientation(p0, *p1, *p2);
50     if (o == 0)
51       return (distSq(p0, *p2) >= distSq(p0, *p1))?
52                    -1 : 1;
53
54     return (o == 2)? -1: 1;
55  }
56
57  void convexHull(Point points[], int n)
58  {
59     int ymin = points[0].y, min = 0;
60     for (int i = 1; i < n; i++)
61     {
62       int y = points[i].y;
63       if ((y < ymin) || (ymin == y &&
64           points[i].x < points[min].x))
65          ymin = points[i].y, min = i;
66     }
67
68     swap(points[0], points[min]);
69
70     p0 = points[0];
71     qsort(&points[1], n-1, sizeof(Point), compare);
72
73     int m = 1; // Initialize size of modified array
74     for (int i=1; i<n; i++)
```

```
    {
        while (i < n-1 && orientation(p0, points[i], points[i+1]) == 0)
           i++;
        points[m] = points[i];
        m++;
    }

    if (m < 3) return;

    stack<Point> S;
    S.push(points[0]);
    S.push(points[1]);
    S.push(points[2]);

    for (int i = 3; i < m; i++)
    {
       while (orientation(nextToTop(S), S.top(), points[i]) != 2)
          S.pop();
       S.push(points[i]);
    }

    while (!S.empty()) // List Of The Points In The Convex Hull
    {
        Point p = S.top();
        cout << "(" << p.x << ", " << p.y <<")" << endl;
        S.pop();
    }
}

int main()
{
    Point points[] = {{0, 3}, {1, 1}, {2, 2}, {4, 4},
                      {0, 0}, {1, 2}, {3, 1}, {3,3}};
    int n = sizeof(points)/sizeof(points[0]);
    /* N is The Number of Points And points is
    the list Of points */
    convexHull(points, n);
    return 0;
}
```

0.3.2 Shoelace Formula (python)

```
def PolygonArea(corners):
    n = len(corners) # of corners
    area = 0.0
    for i in range(n):
        j = (i + 1) % n
        area += corners[i][0] * corners[j][1]
        area -= corners[j][0] * corners[i][1]
    area = abs(area) / 2.0
    return area

# examples
corners = [(2.0, 1.0), (4.0, 5.0), (7.0, 8.0)]
print (PolygonArea(corners))
```

0.3.3 Swap Line

```
#include <bits/stdc++.h>
#define px second
#define py first
typedef pair<long long, long long> pairll;
pairll pnts [MAX];
int compare(pairll a, pairll b)
{
        return a.px<b.px;
```

```
}
double closest_pair(pairll pnts[],int n)
{
        sort(pnts,pnts+n,compare);
        double best=INF;
        set<pairll> box;
        box.insert(pnts[0]);
        int left = 0;
        for (int i=1;i<n;++i)
        {
            while (left<i && pnts[i].px-pnts[left].px > best)
                box.erase(pnts[left++]);
            for(typeof(box.begin()) it=box.lower_bound(make_pair(pnts[i].py-best,
               pnts[i].px-best));it!=box.end() && pnts[i].py+best>=it->py;it++)
                best = min(best, sqrt(pow(pnts[i].py - it->py, 2.0)+pow(pnts[i].px -
                    it->px, 2.0)));
            box.insert(pnts[i]);
        }
        return best;
}
```

0.3.4 Union Of Rectangles

```
#include <bits/stdc++.h>
#define MAX 1000
struct event
{
    int ind;    // Index of rectangle in rects
    bool type; // Type of event: 0 = Lower-left ; 1 = Upper-right
    event(){};
    event(int ind, int type) : ind(ind), type(type){};
};
struct point
{
    int x, y;
};
point rects[MAX][12];
// Each rectangle consists of 2 points: [0] = lower-left ; [1] = upper-right
bool compare_x(event a, event b)
{
    return rects[a.ind][a.type].x < rects[b.ind][b.type].x;
}
bool compare_y(event a, event b)
{
    return rects[a.ind][a.type].y < rects[b.ind][b.type].y;
    }
int union_area(event events_v[], event events_h[], int n, int e)
{
    /*n is the number of rectangles, e=2*n , e is the number of
    points (each rectangle has two points as described in
    declaration of rects)
    */
    bool in_set[MAX] = {0};
    int area = 0;
    sort(events_v, events_v + e, compare_x);
    //Pre-sort of vertical edges
    sort(events_h, events_h + e, compare_y);
    // Pre-sort set of horizontal edges
    in_set[events_v[0].ind] = 1;
    for (int i = 1; i < e; ++i)
    {
        event c = events_v[i];
        int cnt = 0; // Counter to indicate how many
        //rectangles are currently overlapping
        // Delta_x: Distance between current
```

```
43          // sweep line and previous sweep line
44          int delta_x = rects[c.ind][c.type].x - rects[events_v[i - 1].ind]\
45          [events_v[i - 1].type].x;
46          int begin_y;
47          if (delta_x == 0)
48          {
49              in_set[c.ind] = (c.type == 0);
50              continue;
51          }
52          for (int j = 0; j < e; ++j)
53              if (in_set[events_h[j].ind] == 1)
54              {
55                  if (events_h[j].type == 0)
56                  {
57                      if (cnt == 0)
58                          begin_y = rects[events_h[j].ind][0].y;
59                      ++cnt;
60                  }
61                  else
62                  {
63                      --cnt;
64                      if (cnt == 0)
65                      {
66                          int delta_y = (rects[events_h[j].ind][13].y - begin_y);
67                          area += delta_x * delta_y;
68                      }
69                  }
70              }
71          in_set[c.ind] = (c.type == 0);
72      }
73      return area;
74  }
```

0.4 String

0.4.1 LCS

```
1  /* Dynamic Programming C/C++ implementation of LCS problem */
2  #include<bits/stdc++.h>
3
4  int max(int a, int b)
5  {
6      return (a > b)? a : b;
7  }
8
9   int lcs( char *X, char *Y, int m, int n )
10 {
11     int L[m+1][n+1];
12     int i, j;
13
14     for (i=0; i<=m; i++)
15     {
16       for (j=0; j<=n; j++)
17       {
18         if (i == 0 || j == 0)
19           L[i][j] = 0;
20
21         else if (X[i-1] == Y[j-1])
22           L[i][j] = L[i-1][j-1] + 1;
23
24         else
25           L[i][j] = max(L[i-1][j], L[i][j-1]);
26       }
27     }
28     return L[m][n];
```

```
}

int main()
{
  char X[] = "AGGTAB";
  char Y[] = "GXTXAYB";

  int m = strlen(X);
  int n = strlen(Y);

  printf("Length␣of␣LCS␣is␣%dn", lcs( X, Y, m, n ) );

  return 0;
}
```

0.4.2 LIS

```
typedef vector<int> VI;
typedef pair<int,int> PII;
typedef vector<PII> VPII;

#define STRICTLY_INCREASNG

VI LongestIncreasingSubsequence(VI v) {
  VPII best;
  VI dad(v.size(), -1);

  for (int i = 0; i < v.size(); i++) {
#ifdef STRICTLY_INCREASNG
    PII item = make_pair(v[i], 0);
    VPII::iterator iter = lower_bound(best.begin(), best.end(), item);
    item.second = i;
#else
    PII item = make_pair(v[i], i);
    VPII::iterator iter = upper_bound(best.begin(), best.end(), item);
#endif
    if (iter == best.end()) {
      dad[i] = (best.size() == 0 ? -1 : best.back().second);
      best.push_back(item);
    } else {
      dad[i] = dad[iter->second];
      *iter = item;
    }
  }

  VI ret;
  for (int i = best.back().second; i >= 0; i = dad[i])
    ret.push_back(v[i]);
  reverse(ret.begin(), ret.end());
  return ret;
}
```

0.4.3 KMP

```
/*
Finds all occurrences of the pattern string p within the
text string t. Running time is O(n + m), where n and m
are the lengths of p and t, respecitvely.
*/

#include <iostream>
#include <string>
#include <vector>

using namespace std;

```

```
typedef vector<int> VI;

void buildPi(string& p, VI& pi)
{
  pi = VI(p.length());
  int k = -2;
  for(int i = 0; i < p.length(); i++) {
    while(k >= -1 && p[k+1] != p[i])
      k = (k == -1) ? -2 : pi[k];
    pi[i] = ++k;
  }
}

int KMP(string& t, string& p)
{
  VI pi;
  buildPi(p, pi);
  int k = -1;
  for(int i = 0; i < t.length(); i++) {
    while(k >= -1 && p[k+1] != t[i])
      k = (k == -1) ? -2 : pi[k];
    k++;
    if(k == p.length() - 1) {
      // p matches t[i-m+1, ..., i]
      cout << "matched at index " << i-k << ": ";
      cout << t.substr(i-k, p.length()) << endl;
      k = (k == -1) ? -2 : pi[k];
    }
  }
  return 0;
}

int main()
{
  string a = "AABAACAADAABAABA", b = "AABA";
  KMP(a, b); // expected matches at: 0, 9, 12
  return 0;
}
```

0.5 Other

0.5.1 Date Transformation

```
//Dates (C++)

// Routines for performing computations on dates.  In these routines,
// months are exprsesed as integers from 1 to 12, days are expressed
// as integers from 1 to 31, and years are expressed as 4-digit
// integers.

string dayOfWeek[] = {"Mo", "Tu", "We", "Th", "Fr", "Sa", "Su"};

// converts Gregorian date to integer (Julian day number)

int DateToInt (int m, int d, int y){
  return
    1461 * (y + 4800 + (m - 14) / 12) / 4 +
    367 * (m - 2 - (m - 14) / 12 * 12) / 12 -
    3 * ((y + 4900 + (m - 14) / 12) / 100) / 4 +
    d - 32075;
}

// converts integer (Julian day number) to Gregorian date: month/day/year

void IntToDate (int jd, int &m, int &d, int &y){
```

```
  int x, n, i, j;

  x = jd + 68569;
  n = 4 * x / 146097;
  x -= (146097 * n + 3) / 4;
  i = (4000 * (x + 1)) / 1461001;
  x -= 1461 * i / 4 - 31;
  j = 80 * x / 2447;
  d = x - 2447 * j / 80;
  x = j / 11;
  m = j + 2 - 12 * x;
  y = 100 * (n - 49) + i + x;
}

// converts integer (Julian day number) to day of week

string IntToDay (int jd){
  return dayOfWeek[jd % 7];
}
```

0.6 Specific

0.6.1 Max Flow:

```
#include <bits/stdc++.h>
using namespace std;
#define MAXSIZE 502

#define LL long long

// s-> source, d -> sink, n = |V|, e = |E|
int n, e, s, d, c; // s--, d--

vector<int> graph[MAXSIZE];
LL capacity[MAXSIZE][MAXSIZE];
int parent[MAXSIZE];


inline int bfs(vector<int>& parent) {
    fill(begin(parent), end(parent), -1);
    parent[s] = -2;
    queue<pair<int, LL>> q;
    q.emplace(s, __LONG_LONG_MAX__);

    while (!q.empty()){
        int cur = q.front().first;
        auto flow = q.front().second;
        q.pop();

        for (auto v: graph[cur]) {
      if (parent[v] == -1 && capacity[cur][v]) {
        parent[v] = cur;
        LL new_flow = min(flow, capacity[cur][v]);
        if (v == d) return new_flow; // done
        q.emplace(v, new_flow);
      }
        }
    }
    return 0;
}

LL maxFlow(){
  LL flow{0};
  vector<int> parent(n);

```

```
42    while(int new_flow = bfs(parent)) {
43      flow += new_flow;
44      int cur = d;
45      for (int prev; cur != s; cur = prev) {
46        prev = parent[cur];
47        capacity[prev][cur] -= new_flow;
48        capacity[cur][prev] += new_flow;
49      }
50    }
51    return flow;
52  }
```

0.6.2 MCST: Kruskal

```cpp
#include<iostream>
#include<utility>
#include<algorithm>
using namespace std;

typedef pair<int, int>              PII;
typedef pair<int, pair<int, int> > EDGE;

constexpr int maxn = 1e4, maxm = 1e4;

EDGE edge[maxm];
int n, m, mcstSize, mcstCost, rankk[maxn], par[maxn];

int ds_find(int u){
    if(par[u] == u) return u;
    return par[u] = ds_find(par[u]);
}

void ds_union(int u, int v){
    int uu = ds_find(u),vv = ds_find(v);
    if(rankk[uu] == rankk[vv]){
        ++rankk[uu];
        par[uu] = vv;
    }
    else if(rankk[uu] < rankk[vv])
        par[vv] = uu;
    else // same height, select one at random
        par[uu] = vv;
}

void kruskal(){
    generate(par, par+n, [n=0]() mutable {return n++;});
    sort(edge, edge+m);
    for(int i=0; i<m && mcstSize<n-1; i++){
        int u = edge[i].second.first, v = edge[i].second.second;
        if(ds_find(u) != ds_find(v)){
            ds_union(u, v);
            mcstCost += edge[i].first;
            ++mcstSize;
        }
    }
}

int main()
{
    cin >> n>> m;
    for(int i=0; i<m; i++){
        cin >> edge[i].second.first
            >> edge[i].second.second
            >> edge[i].first;
        edge[i].second.first--,edge[i].second.second--;
    }


    kruskal();

    cout << mcstCost << endl;
}
```

0.6.3 Articulation Point

```cpp
#include <bits/stdc++.h>
using namespace std;
#define MaxSize 10001

vector <int> graph[MaxSize];
bool visited[MaxSize];
vector<int> art_point;
int low[MaxSize],disc[MaxSize],parent[MaxSize];
int timeii = 0;

void dfs_art_point(int v) {
    int ch = 0; visited[v] = true;
    disc[v] = low[v] = ++timeii;
    for (auto c: graph[v]) {
        if (!visited[c]) {
            ch++;
            parent[c] = v;
            dfs_art_point(c);
            low[v] = min(low[c], low[v]);

            if (parent[v] == -1 && ch > 1)art_point.push_back(v);
            if (parent[v] != -1 && low[c]>=disc[v])art_point.push_back(v);
        }
        else if (c != parent[v])
            low[v] = min(low[v], disc[c]);
    }
}

int main() {
    int n, m; cin >> n >> m;

    fill(parent, parent+n+2, -1);
    fill(disc, disc+n+2, -1);
    fill(low, low+n+2, -1);

    for (int j = 0;j<m; j++) {
        int a, b; cin >> a >> b;
        graph[a].push_back(b);
        graph[b].push_back(a);
    }
    for (int j = 1; j <= n; j++)
        if (!visited[j])dfs_art_point(j);
    for(auto a: art_point)cout << a << "␣";
}
```