Estructura de Datos y Algoritmia

Misericordia Avila Irigaray

2018/19

MERGE SORT

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

void merge(vector<int>& arr, int l, int m, int r){
    int i, j, k;
    int n1 = m - l + 1;
    int n2 =  r - m;

    /* create temp arrays */
    vector<int> L(n1);
    vector<int> R(n2);

    /* Copy data to temp arrays L[] and R[] */
    for (i = 0; i < n1; i++)
        L[i] = arr[l + i];
    for (j = 0; j < n2; j++)
        R[j] = arr[m + 1+ j];

    /* Merge the temp arrays back into arr[l..r]*/
    i = 0; // Initial index of first subarray
    j = 0; // Initial index of second subarray
    k = l; // Initial index of merged subarray
    while (i < n1 and j < n2){
        if (L[i] <= R[j]) {
            arr[k] = L[i];
            i++;
        }
        else{
            arr[k] = R[j];
            j++;
        }
        k++;
    }

    /* Copy the remaining elements of L[], if there
       are any */
    while (i < n1){
        arr[k] = L[i];
        i++;
        k++;
    }

    /* Copy the remaining elements of R[], if there
       are any */
    while (j < n2){
        arr[k] = R[j];
        j++;
        k++;
    }
}

/* l is for left index and r is right index of the
   sub-array of arr to be sorted */
void mergeSort(vector<int>& arr, int l, int r){
    if (l < r){
        // Same as (l+r)/2, but avoids overflow for
        // large l and h
        int m = l+(r-l)/2;

        // Sort first and second halves
        mergeSort(arr, l, m);
        mergeSort(arr, m+1, r);

        merge(arr, l, m, r);
    }
}
```

QUICK SORT

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


int partition (vector<int>& T, int e, int d){
  int pivot = T[e];
  int i = e;
  int j = d;
  while(i<j){
    while (T[i]<=pivot) ++i;
    while (T[j] > pivot) --j;
    if (i < j) swap(T[i], T[j]);
  }
  swap(T[e], T[j]);
  return j;
}


void quickSort(vector<int>& a,int l,int u){
    if(l<u){
        int j = partition(a,l,u);
        quickSort(a,l,j);
        quickSort(a,j+1,u);
    }
}

int main(){
  int n, x;
  cin >> n;
  vector<int> v(n);
  int i = 0;
  while(i < n){
    cin >> x;
    v[i]=x;
    ++i;
  }
  i = 1;
  quickSort(v,0,n);
  while(i <= n){
    if(i==n) cout << v[i] << endl;
    else cout << v[i] << ' ';
    ++i;
  }
}
```

MODULAR EXPONENTIATION

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


int modular_exponentation(int n, int k, int m) {
   if (k == 0) return 1;
   if (k % 2 != 0) return (n%m*modular_exponentation(n,k-1,m))%m;
   int ret = modular_exponentation(n,k/2,m)%m;
   return (ret*ret)%m;
}

int main() {
   int n, k, m;
   while (cin >> n >> k >> m) {
            cout << modular_exponentation(n,k,m) << endl;
   }
}
```

DFS

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

typedef vector<vector<int>> Graph;

// Devuelve si la c.c. de i es ciclica
bool ciclico(const Graph& G, int x, vector<bool>& visitado)
{
   stack<pair<int, int>> DFS;
   DFS.push({x, -1});
   while (!DFS.empty()) {
      int v = DFS.top().first;
      int padre = DFS.top().second;
      DFS.pop();
      visitado[v] = true;
      for (int s : G[v]) {
         if (s == padre) continue;
         else if (visitado[s]) return true;
         else DFS.push({s, v});
      }
   }

   return false;
}

void bosc(const Graph& G)
{
   int n = G.size();
```

```
    vector<bool> visitado(n, false);
    int s = 0;
    for (int i = 0; i < n; ++i) {
        if (visitado[i]) continue;
        if (ciclico(G, i, visitado)) {
            cout << "no" << endl;
            return;
        }
        ++s;
    }
    cout << s << endl;
}

int main()
{
    int n, m;
    while (cin >> n >> m) {
        Graph G(n);
        for (int i = 0; i < m; ++i) {
            int x, y;
            cin >> x >> y;
            G[x].push_back(y);
            G[y].push_back(x);
        }
        bosc(G);
    }
}
```

BFS

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

vector<pair<int, int>> direcs = { {0,1}, {0,-1}, {1,0}, {-1,0} };

int main()
{
   int n, m;
   cin >> n >> m;
   vector<vector<char>> M(n, vector<char>(m));
   for (int i = 0; i < n; ++i)
      for (int j = 0; j < m; ++j)
         cin >> M[i][j];
   int y, x;
   cin >> y >> x;
   --y; --x;
   vector<vector<int>> dist(n, vector<int>(m, -1));
   dist[y][x] = 0;
   queue<pair<int, int>> Q;
   Q.push({y, x});
   int cuantos = -1;
   while (not Q.empty()) {
      int i = Q.front().first;
      int j = Q.front().second;
      Q.pop();
      if (M[i][j] == 't') {
            cuantos = dist[i][j];
        }
        for (const auto &direc : direcs) {
            int I = i + direc.first;
            int J = j + direc.second;
            if (I >= 0 and J >= 0 and I < n and J < m and M[I][J] != 'X' and dist[I][J] == -1) {
                dist[I][J] = dist[i][j] + 1;
                Q.push({I, J});
            }
        }
    }
    if (cuantos == -1)
        cout << "no es pot arribar a cap tresor" << endl;
    else
        cout << "distancia maxima: " << cuantos << endl;
}
```

DIJKSTRA’S ALGORITHM

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

const int MAX = 10000000;

typedef pair<double, int> WArc;
typedef vector<vector<WArc>> WGraph;

int dijkstra(const WGraph& G, int s, int goal)
{
   int n = G.size();
   vector<double> d(n, MAX);
   d[s] = 0;
   vector<int> p(n, -1);
   vector<bool> S(n, false);
   priority_queue<WArc, vector<WArc>, greater<WArc>> Q;
   Q.push(WArc(0, s));
   while (not Q.empty()) {
      int u = Q.top().second; Q.pop();
      if (not S[u]) {
         S[u] = true;
         for (WArc a : G[u]) {
            int v = a.second;
            double c = a.first;
            if (d[v] > d[u] + c) {
               d[v] = d[u] + c;
               p[v] = u;
               Q.push(WArc(d[v], v));
            }
         }
      }
   }
   return d[goal];
}

int main()
{
   int n, m;
   while (cin >> n >> m) {
      WGraph G(n);
      for (int i = 0; i < m; ++i) {
         int u, v, c;
         cin >> u >> v >> c;
         G[u].push_back({c, v});
      }
      int x, y;
      cin >> x >> y;
      int coste = dijkstra(G, x, y);
      if (coste == MAX) {
         cout << "no path from " << x << " to " << y << endl;
      } else {
         cout << coste << endl;
      }
   }
}
```

TOPOLOGICAL SORT

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

int main()
{
    int n, m;
    while (cin >> n >> m) {
        vector<vector<int>> G(n);
        vector<int> gent(n);
        for (int i = 0; i < m; ++i) {
            int x, y;
            cin >> x >> y;
            G[x].push_back(y);
            ++gent[y];
        }
        priority_queue<int, vector<int>, greater<int>> prio;
        for (int i = 0; i < n; ++i) {
            if (gent[i] == 0) prio.push(i);
        }
        bool primer = true;
        while (not prio.empty()) {
            int u = prio.top(); prio.pop();
            if (primer) primer = false;
            else cout << " ";
            cout << u;
            for (int v : G[u]) {
                if (--gent[v] == 0) prio.push(v);
            }
        }
        cout << endl;
    }
}
```