C++ Graph Theory Sample Cheat Sheet

by Hackin7 via cheatography.com/71996/cs/20334/

Representation

```
///Adjacency
Matrix//////////////////
int V, E, A, B, W, g[1005 ][1 -
005];
cin >> V >> E; memset(g, -1,
sizeof (g));
for (int i = 0; i < E; i++) {
       cin >> A >> B >> W;
        //W eight, can set for
both or single direction
        g[A][B] = W;
        g[B][A] = W;
}
///Adj acency
List// /// /// /// /// /// /////
vector <pa ir<int, int> >
g[1005];
int V, E, A, B, W;
cin >> V >> E;
for (int i = 0; i < E; i++) {
       cin >> A >> B >> W;
        g[A ].p ush _ba ck( -
mak e_p air(B, W));
        g[B ].p ush _ba ck( -
mak e_p air(A, W));
}
```

Floyd-Warshall

```
//initialise dist[i][j] to
infinity at the start
for (int k=0;k< n;k++)
       for (int i=0;i< n;i++)
               for (int j=0;j< -
n;j++)
```

Floyd-Warshall (cont)

```
>       // if there is a shorter path through
node k, take it!
         dist[i][j] = min(dist[i][j], dist[i][k]+dist-
[k][j]);w
```

Floyd-Warshall algorithm uses the idea of triangle inequality, and is very
easy to code (just 4 lines!)

If there are negative cycles, dist[i][i] will be negative. Note the order!!!

Prim's Algorithm

```
//Lol just copied from
hackerearth website
#include <io str eam>
#include <ve cto r>
#include <qu eue>
#include <fu nct ion al>
#include <ut ili ty>
using namespace std;
const int MAX = 1e4 + 5;
typedef pair<long long, int>
PII;
bool marked [MAX];
vector <PI I> adj[MAX];
long long prim(int x)
{
        pri ori ty_ que ue<PII,
vector <PI I>, greate r<P II> >
Q;
       int y;
```

Prim's Algorithm (cont)

```
>    long long minimumCost = 0;
   PII p;
   Q.push(make_pair(0, x));
   while(!Q.empty())
   {
      // Select the edge with minimum
weight
      p = Q.top();
      Q.pop();
      x = p.second;
      // Checking for cycle
      if(marked[x] == true)
         continue;
      minimumCost += p.first;
      marked[x] = true;
      for(int i = 0;i < adj[x].size();++i)
      {
         y = adj[x][i].second;
         if(marked[y] == false)
            Q.push(adj[x][i]);
      }
   }
   return minimumCost;
}
int main()
```

By **Hackin7**
cheatography.com/hackin7/

Published 21st August, 2019.
Last updated 27th December, 2019.

Sponsored by **Readable.com**
Measure your website readability!
https://readable.com

Prim's Algorithm (cont)

```
> {
    int nodes, edges, x, y;
    long long weight, minimumCost;
    cin >> nodes >> edges;
    for(int i = 0;i < edges;++i)
    {
        cin >> x >> y >> weight;
        adj[x].push_back(make_pair(weight,
y));
        adj[y].push_back(make_pair(weight,
x));
    }
    // Selecting 1 as the starting node
    minimumCost = prim(1);
    cout << minimumCost << endl;
    return 0;
}
```

Used to Construct MST from Graph

Lowest Common Ancestor of Tree

```
ll lca(ll N,ll a,ll b){
        if( dep th[ a]< dep -
th[b]) swap(a,b);
        //E qualise depth
        for(ll k=log2 (N) ;k> -
=0; k--){
                ll parent =
find_p are nt( a,k );/ /p[ a][k]
                 if( par ent!=-1
&& depth[ par ent ]>= dep th[b])
{
                        a=p -
arent;
```

Lowest Common Ancestor of Tree (cont)

```
>      }
    }
    if (a==b)return a;
    //Jump parent by parent
    for(ll k=log2(N);k>=0;k--){
        ll parent = find_parent(a,k);//p[a][k]
        ll parentb = find_parent(b,k);//p[b][k]
        if(parent!=parentb)a=parent,b=parentb;
    }
    return p[a][0];
}
```

Requires 2k Decomposition of Parents

Breadth First Search

```
vector<int> g[100005];
queue< pai r<int, int> > q;
int dist[1 000 005];
fill(dist, dist+1 000005, -1);
while (!q.em pty()) {
        int v = q.fron t().first;
        int d = q.fron t().se -
cond;
         q.p op();
        if (dist[v] != -1)
continue; //Visited
         dist[v] = d;
        for (int i = 0; i <
g[v].s ize(); i++) {
```

Breadth First Search (cont)

```
>       q.push(make_pair(g[v][i], d+1));
    }
}
```

Time Complexity: O(|V| + |E|)
Space Complexity: O(b^d)
where d is the depth of the graph and b is the branching factor.

BFS is more suitable when the goal is close to the source, BFS is still faster in such cases.

We can use this algorithm to find the shortest path in a grid/unweighted graph

Bellman-Ford

```
dist[s]=0; //dist all others =
INF
for (int i=0; i<N-1; i++){
        for (int j=0; j<E; j++){
                // if path is
shorter through node u, take it!
                 dist[v] =
min(di st[v], dist[u ]+c ost);
        }
}
```

Solves the Single Source Shortest Path (SSSP) problem. (shortest path from one node (source) to all other nodes)
Can be used with negative edges, Run the algorithm twice to detect for negative cycles

Time Complexity: O(VE)
Space Complexity: O(V)

Sponsored by **Readable.com**
Measure your website readability!
https://readable.com

Union Find Data Structure

```
int root (int x ) {
        if (x == parent [x])
return x ;
         return root (paren t[x])
;
}
bool is_con nected (int x,int y)
{
         return root (x) ==
root(y) ;
}
void connect ( int x , int y ) {
        int root_x = root (x);
        int root_y = root (y);
        if (root_x != root_y)
                 parent [root_x]
= root_y ;
}
////For Rankin -
g// /// /// /// /// /// /////
int rank[N];
void connect (int x , int y) {
        int root_x = root (x) ,
root_y = root (y) ;
        if (root_x == root_y)
return ; // same root
        if (rank[ root_x] >
rank[r oot_y]) {
                 par ent [ro -
ot_y] = root_x ;
        } else if (rank[ root_x]
< rank[r oot_y]) {
                 par ent [ro -
ot_x] = root_y ;
        } else {
                 par ent [ro -
ot_y] = root_x ;
```

Union Find Data Structure (cont)

```
>        rank[root_x]++;
  }
}
```

Kruskal's Algorithm for MST

```
vector <tuple<int,int,int> >
edges ; // weight,node A,node B
sort (edges.be gin(), edges.end
()) ;
int total_ weight = 0;
for (auto e : edges) {
        int weight, a, b;
        tie (weigh t,a,b) = e ;
        if (root(a) == root(b))
// taking this edge will cause a
cycle
                 con tinue;
         tot al_ weight += weight
; // take this edge
         connect (a, b) ; //
connect them in the UFDS
}
```

Sort the list of edges by weight
For each edge in ascending order: If both nodes aren't already
connected, take it. Else, skip this edge.
Time complexity: O(E log V) (but faster than Prim's algorithm in
practice)
UFDS is needed to check if the nodes are connected in (2).

Depth First Search

```
bool vis[N];
vector <in t> adjLis t[N];
void dfs(int node) {
        if (vis[n ode]) return;
         vis [node] = true;
        for (int a = 0; a <
(int)a djL ist [no de].si ze();
++a)
                 dfs (ad jLi -
st[ nod e][a]);
}
///Ite rat -
ive /// /// /// /// /// /// /// -
/// /// /////
bool vis[N];
vector <in t> adjLis t[N];
stack< int> S;
while (!S.em pty()) {
        int node = S.top();
         S.p op();
        if (vis[n ode]) continue;
         vis [node] = true;
        for (int a = 0; a <
(int)a djL ist [no de].si ze();
++a)
                 S.p ush (ad -
jLi st[ nod e][a]);
}
```

DFS uses O(d) space, where d is the depth of the graph

DFS is not suited for infinite graphs.

Some applications of DFS include:
1. Topological Ordering (covered later)
2. Pre-/In-/Post-order numbering of a tree
3. Graph connectivity
4. Finding articulation points
5. Finding bridges

Sponsored by **Readable.com**
Measure your website readability!
https://readable.com

Dijkstra's Algorithm

```
vector<pair<int,int> >
adjList[10000]; // node, weight
priori ty_ que ue< pai r<i nt, -
int >, vector <pa ir< int ,in t>
>, greate r<p air <in t,i nt> >
> pq; // distance, node
int dist[1 0000];
memset (dist, -1, sizeof (di -
st));
pq.pus h(m ake _pa ir(0,
source)); dist[0] = 0;
while( !pq.em pty()){
        pai r<i nt, int> c =
pq.top();
        pq.p op();
        if( c.first !=
dist[c.se cond]) continue;
        for (auto i : adjLis -
t[c.se cond]){
                if( dis -
t[i.first] == -1 ||
dist[i.first] > c.first +
i.second){
                        dis -
t[i.first] = c.first + i.second;
                        pq.p -
us h(m ake _pa ir( dis t[i.fi -
rst], i.first));
                }
       }
}
```

Time Complexity of our implementation: O(E log E)
Space Complexity: O(V+E)

Solves the Single Source Shortest Path (SSSP) problem. Means shortest path from one node to all other nodes. Cannot be used with negative edges as it runs too slow
Especially cannot be used with negative cycles

2k Parent Decomposition

```
typedef long long ll;
ll p[V][K]; //node,kth ancestor
//DFS to compute node parents
for p[i][0], first parent
bool visite d[V];
ll depth[V];
void dfs(ll x){
       if (visit ed[ x]) return;
        vis ite d[x ]=true;
       for (auto i:adjl ist [x])
{
               if (!visi ted -
[i.f ir st]){
                       if
(p[i.f irs t][0] == -1){
                               -
//c out <<i.fi rst <<"< -"<< -
x<< endl;
                               -
p[i.fi rst][0] = x;
                               -
dep th[ i.f irst] = depth[ x]+1;
                       }
                        dfs -
(i.f irst);
               }
       }
}
void calc_k _pa ren ts(ll N){
       for (ll k=1;k< K;k++){
               for (ll i=0;i< -
N;i++){
                       if (p[i]
[k-1] != -1){
```

2k Parent Decomposition (cont)

```
>              p[i][k]= p[p[i][k-1]][k-1];
        }else{p[i][k]=-1;}
       // if (k==2)cout<<i<<","<<k<<":"<<-
p[i][k-1]<<","<<p[p[i][k-1]][k-1]<<","<<p[i][k]
<<endl;
      }
   }
}
ll find_parent(ll x,ll k){
   for (ll i=K;i>=0;i--){
      if (k>= (1<<i)){
         if (x==-1)return -1;
         x=p[x][i];
         k-=1<<i;
      }
   }
   return x;
}
```


Sponsored by **Readable.com**
Measure your website readability!
https://readable.com