Minimum Spanning Trees

Problem Solving Club
January 25, 2017
A tree is an **undirected** graph. The following are all equivalent definitions:

- Any two vertices are connected by exactly one path
- Connected with exactly V-1 edges
- Connected and has no cycles
A spanning tree of an undirected graph $G$ is a tree that includes all vertices of $G$.

- Does every graph have a spanning tree?
- Can a graph have more than one spanning tree?
  - The number of spanning trees of any graph can be found using Kirchhoff’s theorem
  - Take the determinant of a $V \times V$ matrix, where the entry in row $i$ and column $j$ is:
    - The degree of vertex $i$, if $i = j$
    - $-1$, if vertices $i$ and $j$ are adjacent
    - $0$, otherwise
Minimum spanning trees

A **minimum spanning tree** is a spanning tree with the **minimum total edge weight**.

What are some practical applications for MST?
- The first MST algorithm was invented in 1926 to find an efficient electrical grid.
- Design of computer networks.
- Cluster analysis.
Disjoint-set (union-find) data structure

- Keeps track of a set of objects partitioned into disjoint subsets.
- Supports two operations:
  - Find: Determine which subset an object is in.
  - Union: Union two subsets.
- It is possible to implement the operations in effectively constant time (inverse of Ackermann function).
- In programming contests, usually copy the (short) code from somewhere.
Kruskal’s algorithm is a greedy algorithm that finds a minimum spanning tree.

- Sort edges by ascending weight.
- While the tree is not complete:
  - Choose an edge with the lowest weight that has not been chosen yet.
  - Add the edge if it connects two different connected components.
- How to find a maximum spanning tree?
Example code for Kruskal’s algorithm

```cpp
bool edge_cmp(const edge &a, const edge &b) {
    return a.weight < b.weight;
}

vector<edge> mst(int n, vector<edge> edges) {
    union_find uf(n);
    sort(edges.begin(), edges.end(), edge_cmp);
    vector<edge> res;
    for (int i = 0; i < edges.size(); i++) {
        int u = edges[i].u, v = edges[i].v;
        if (uf.find(u) != uf.find(v)) {
            uf.unite(u, v);
            res.push_back(edges[i]);
        }
    }
    return res;
}
```

// Disjoint set data structure O(log n)

```cpp
#define MAXN 1000
int p[MAXN];
int find(int x) {
    return p[x] == x ? x : p[x] = find(p[x]);
}
void unite(int x, int y) {
    p[find(x)] = find(y);
}
for (int i = 0; i < MAXN; i++) p[i] = i;
```