Greedy algorithms

- take whatever you can get!
- Making change

```
Set change( n ) {
    const Set coins = { 100, 25, 10, 5, 1 };
    int sum = 0;
    while( ! sum == n ) {
        x = largest item in coins so that sum+x<=n
        if( no x ) {
            return error( “no solution exists” );
        }
        S = union( S, {x} );
        sum += x;
    }
}
```

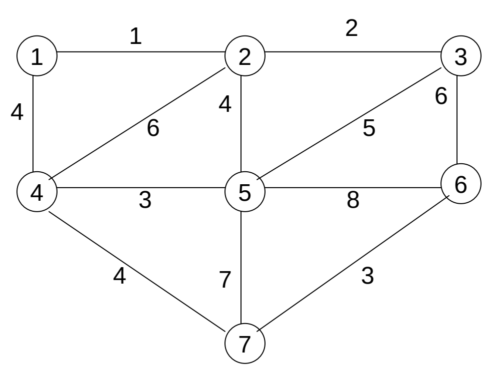

Structure of a Greedy Algorithm

```
Set greedy( Set candidates ) {
    Set result = EMPTYSET;
    while( ! candidates. isEmpty() && ! isSolution( result ) ) {
        Object x = select( candidates );
        candidates. remove( x );
        if( feasible( union( result , x ) ) {
            result. add( x );
        }
    }
    if( isSolution( result ) ) {
        return result ;
    } else {
        error( “there are no solutions” );
    }
}
```

Minimum Spanning Trees

- Problem:
  - we have a graph
    - edges have a cost
  - we need a spanning tree that's cheap
  - number of edges in a spanning tree?
  - sample application: creating a network to connect cities
- set of candidates?
- isSolution function?
- isFeasible function?

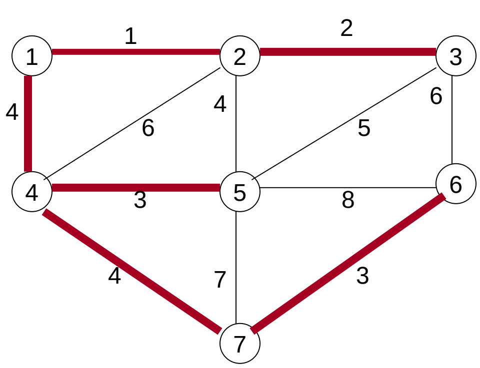
Kruskal's Algorithm: Example



Kruskal's Algorithm

1. make a partition for each node
2. repeat until one partition left:
   1. pick shortest remaining edge
   2. if it is inside one partition: reject
      else:
      1. merge the two partitions
      2. add edge to solution set
3. done.

Kruskal's Algorithm: Example



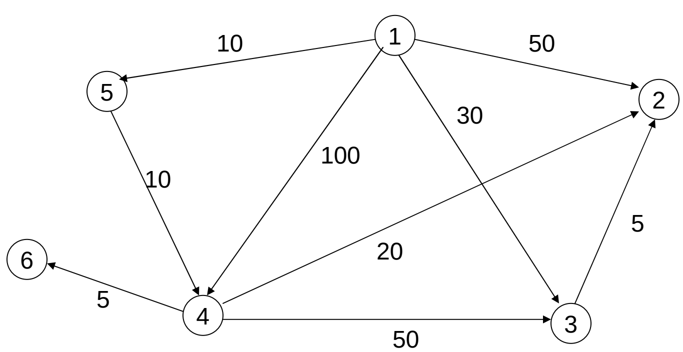
Kruskal's Algorithm

```
Edge[] minimumSpanningTree( Graph g ) {
    int e = g. getNumberOfEdges();
    int n = g. getNumberOfNodes();
    Edge[] mst = new Edge[ n-1 ];
    UnionFind uf = new UnionFind( n );
    Edge[] candidates = g. getEdges();
    sort( candidates );
    for( int i = 0, k = 0;  i < e  &&  k < n-1; i++ ) {
        Set set1 = uf. find( candidates[ i ]. firstNode() );
        Set set2 = uf. find( candidates[ i ]. secondNode() );
        if( set1 != set2 ) {
            uf. unite( set1, set2 );
            mst[ k++ ] = candidates[ i ];
        }
    }
    return mst;
}
```

- Runtime: O(e log n)
  - e: edges, n: nodes

Shortest Paths: Dijkstra's Algorithm

- How cheap (fast) can you get from one node to all others?
  - directed graph with cost
  - sample application: distances from one city to other cities



Dijkstra’s Algorithm

- candidates = set of nodes
- result = array of (preliminary) distances to start node for all other nodes
- in each step:
  - pick the candidate that has shortest distance to start node so far
  - check if it makes the distance to any other node shorter
  - if so, update distances

Dijkstra’s Algorithm: Example

step	node considered	remain. candid.	result[2]	result[3]	result[4]	result[5]	result[6]
1							
2							
3							
4							
5							

Dijkstra's Algorithm

```
int[] shortestPath( graph ) {                          // node 1: start node
    Set candidates = graph. getSetofNodes();
    candidates. remove( 1 );
    int[] result = new int[];
    for( i = 2; i <= n; i++ ) {
        result[ i ] =  graph. getCost( 1, i );        // may be ∞
    }
    while( candidates. size() > 0 ) {
        int node = element of candidates so that result[ node ] minimal
        candidates. remove( node );
        for( int c: candidates ){
            result[ c ] = min( result[ c ], result[ node ] + graph. getCost(
              node, c )
        }

    }
    return result;
}
```

- Runtime: O(n^2)
  - O(e log n) if inverted heap is used for candidate nodes (which is better?)