```
------------------------------------------------------------------
graph_maxflow.cpp: compute the max flow for a weighted, directed
graph using DFS (Ford-Fulkerson) or BFS (Edmunds-Karp) aug path
------------------------------------------------------------------

#include <...>
using namespace std;

template <typename Tkey, typename Twgt>
class graph {

  see graph_wgt.cpp for basic definitions

  public:
    void maxflow(Tkey &, Tkey &);

  private:
    void maxflow(int, int);
    bool augpath_xxx(int,int);  -- xxx = dfs or bfs

    typedef enum { WHITE, BLACK } vcolor_t;
    vector<vcolor_t> vcolor;
    vector<int> vlink;

    vector< vector<Twgt> > capacity;
    vector< vector<Twgt> > flow;
};

template <typename Tkey, typename Twgt>
void graph<Tkey,Twgt>::maxflow(Tkey &source_key, TKey &sink_key) {

  do the usual error checking

  maxflow(key_map[source_key], key_map[sink_key]);
}

int main(int argc, char *argv[]) { ... }
  if (argc != 4) {
    cerr << "usage: " << argv[0]
         << " source sink graph.txt\n";
  }

  string source = argv[1];
  string sink = argv[2];
  const char *fname = argv[3];

  graph<string,int> G;
  G.read(fname);
  G.maxflow(source, sink);
}
```

```
template <typename Tkey, typename Twgt>
void graph<Tkey,Twgt>::maxflow(int source, int sink) {
  int N = V.size();
  capacity.assign(N, vector<Twgt>(N, 0));
  flow.assign(N, vector<Twgt>(N, 0));

  for (int i=0; i<(int)V.size(); i++) {
    for (int k=0; k<(int)E[i].size(); k++) {
      int j = E[i][k];
      int wgt = W[i][k];
      capacity[i][j] = wgt;
    }
  }

  while (augpath_xxx(source, sink)) {
    stack<Tkey> path;

    Twgt delta = numeric_limits<Twgt>::max();
    for (int j=sink; j!=source; j=vlink[j]) {
      path.push(V[j]);

      int i = vlink[j];
      if (delta > capacity[i][j] - flow[i][j])
        delta = capacity[i][j] - flow[i][j];
    }
    path.push(V[source]);

    while (!path.empty()) {
      cout << path.top() << " ";
      path.pop();
    }
    cout << " = " << delta << "\n";

    for (int j=sink; j!=source; j=vlink[j]) {
      int i = vlink[j];
      flow[i][j] += delta;
      flow[j][i] -= delta;
    }
  }

  Twgt max_flow = 0;
  for (int j=0; j<(int)V.size(); j++) {
    if (flow[source][j] > 0)
      max_flow += flow[source][j];
  }

  cout << "MAXFLOW = " << max_flow << "\n";
}
```

```
template <typename Tkey, typename Twgt>
bool graph<Tkey,Twgt>::augpath_dfs(int source, int sink) {
  vcolor.assign(V.size(), WHITE);
  vlink.assign(V.size(), -1);

  stack<int> S;
  S.push(source);

  while (!S.empty()) {
    int i=S.top();
    S.pop();




    vcolor[i] = BLACK;
    if (i == sink)
      break;

    for (int j=V.size()-1; 0<=j; j--) {
      if (vcolor[j] == BLACK)
        continue;

      if (flow[i][j] < capacity[i][j]) {
        vlink[j] = i;
        S.push(j);
      }
    }
  }

  while (!S.empty())
    S.pop();

  return vcolor[sink] == BLACK;
}
```

```
template <typename Tkey, typename Twgt>
bool graph<Tkey,Twgt>::augpath_bfs(int source, int sink) {
  vcolor.assign(V.size(), WHITE);
  vlink.assign(V.size(), -1);

  queue<int> Q;
  Q.push(source);

  while (!Q.empty()) {
    int i=Q.front();
    Q.pop();

    if (vcolor[i] == BLACK)
     continue;

    vcolor[i] = BLACK;
    if (i == sink)
      break;

    for (int j=0; j<(int)V.size(); j++) {
      if (vcolor[j] == BLACK || vlink[j] = -1)
        continue;

      if (flow[i][j] < capacity[i][j]) {
        vlink[j] = i;
        Q.push(j);
      }
    }
  }

  while (!Q.empty())
    Q.pop();

  return vcolor[sink] == BLACK;
}
```