Implementazione di MinCut MonteCarlo – Cattaneo Kevin - S4944382

Per il seguente esercizio della ricerca del taglio minimo all'interno di un grafo si è utilizzato il linguaggio di programmazione C++ per una questione di comodità sul controllo e gestione dei grafi, per cui si è utilizzata una matrice di due vettori.

Codice C++ di seguito:

```
#include <iostream>
#include <vector>
#include <random>

using Graph = std::vector<std::vector<int>>;
const int DIM = 9; // Dimensione pre-impostata sui vector

// Riferimento al manuale per cercare una distribuzione uniforme
std::random_device rand_dev;
std::mt19937 generator(rand_dev());
std::uniform_int_distribution<int>  distribution(0,8);

// Funzione di stampa del grafo
void printGraph(const Graph& g){
  std::cout << "\n     ";
  for(int i=0; i<g.size(); i++) std::cout << i << " -- ";
  std::cout << std::endl;
  for(int i=0; i<g.size(); i++){
    std::cout << i << "| ";
    for(int j=0; j<g.size(); j++)
      std::cout << "  " << g[i][j] << "  ";
    std::cout << std::endl;
  }
  std::cout << std::endl;
}

// Funzione di ricerca di taglio minimo (di un grafo candidato)
// Nota: l'input non deve avere cappi
int MC_MinCut(const Graph& g){
  Graph auxG = g;
  std::vector<bool> exists(DIM); // vettore ausiliario per 'salvarsi' i nodi esiste
nti
  for(int i=0; i<DIM; i++) exists[i] = 1;

  for(int i=auxG.size()-1; i>=2; --i){
    //1. campiona un arco G con probabilita uniforme e identifica i suoi vertici, u
 e v, in un nuovo vertice uv
    int u,v;
    //Scelgo un arco esistente
    do{
      u = distribution(generator);
      v = distribution(generator);
    }while(!exists[u] || !exists[v] || !auxG[u][v]);

    //2. rimuovi tutti gli archi che univano i vertici u e v, incluso quello campio
nato, e mantieni tutti gli archi che incidono sul nuovo vertice uv
```

```
    // Collasso v su u creando uv (rimane u 'fuso')
    for(int j=0; j<g.size(); ++j){
      auxG[u][j] += auxG[v][j];
      //auxG[v][j] = 0; // risparmio operazioni
      auxG[j][u] += auxG[j][v];
      auxG[j][v] = 0;
    }
    auxG[u][v] = auxG[v][u] = auxG[u][u] = 0;
    exists[v] = 0;
  }

  // Il grafo risultante auxG è costituito dagli archi che uniscono gli ultimi due
vertici rimasti del grafo g
  int aux = -1;
  for(int i=0; i<auxG.size(); i++)
    if(exists[i]){
      aux = i;
      break;
    }

  int mincut = 0;
  for(int i=0; i<auxG.size(); ++i)
    if(auxG[aux][i] > 0)
      mincut += auxG[aux][i];
  return mincut;
}


int main(){

  Graph g(DIM, std::vector<int>());
  // Grafo di Fritsch
  //{0,1},{0,2},{0,4},{0,5},{0,8},{1,2},{1,3},{1,7},{1,8},{2,3},{2,4},
    //{3,4},{3,6},{3,7},{4,5},{4,6},{5,6},{5,8},{6,7},{6,8},{7,8}
        // Assegno zero a ogni valore di arco (così da allocare memoria)
  for(int i=0; i<g.size(); i++)
    g[i].assign(DIM,0);

  // Riempimento grafo di Fritsch
  g[0][1] = 1;
  g[1][0] = 1;
  g[0][4] = 1;
  g[4][0] = 1;
  g[0][5] = 1;
  g[5][0] = 1;
  g[0][8] = 1;
  g[8][0] = 1;
  g[0][2] = 1;
  g[2][0] = 1;
  g[1][2] = 1;
  g[2][1] = 1;
  g[1][3] = 1;
  g[3][1] = 1;
```

```
  g[1][7] = 1;
  g[7][1] = 1;
  g[1][8] = 1;
  g[8][1] = 1;
  g[2][3] = 1;
  g[3][2] = 1;
  g[2][4] = 1;
  g[4][2] = 1;
  g[3][4] = 1;
  g[4][3] = 1;
  g[3][6] = 1;
  g[6][3] = 1;
  g[3][7] = 1;
  g[7][3] = 1;
  g[4][5] = 1;
  g[5][4] = 1;
  g[4][6] = 1;
  g[6][4] = 1;
  g[5][6] = 1;
  g[6][5] = 1;
  g[5][8] = 1;
  g[8][5] = 1;
  g[6][7] = 1;
  g[7][6] = 1;
  g[6][8] = 1;
  g[8][6] = 1;
  g[7][8] = 1;
  g[8][7] = 1;

  printGraph(g);

  // Elaborazione statistiche
  int f = 0;
  for(int i=0; i<100000; i++)
    if(MC_MinCut(g) == 4) f++;
  std::cout << "Con una frequenza empirica di " << f << " ottengo un taglio minimo"
 << std::endl;

  double prob = 1.0 * f/100000;
  std::cout << "Probabilità di ottenere un taglio minimo: " << prob << std::endl;

  // Poiché la probabilità di ottenere almeno un taglio in n lanci è uguale al
complemento a 1 della probabilità di ottenere n volte il contrario, scriviamo (1 -
p)^n = 10-3, su cui, applicando il log naturale, ottengo n = -6.9 circa
  int run = ceil(-6.9/log(1-prob));
  std::cout << "Run necessarie per ottenere almeno un taglio minimo con una probabi
lità del 99,9%: " << run << std::endl;
}
```

Immagine del grafo utilizzato e delle corrispondenti label assegnate ad ogni nodo:

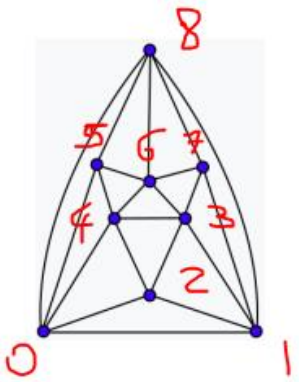

Figura 4: Il grafo di *Fritsch* ha $n = 9$ vertici e 21 archi.

Screenshot dell'output ottenuto:

```
     0 -- 1 -- 2 -- 3 -- 4 -- 5 -- 6 -- 7 -- 8 --
0|   0    1    1    0    1    1    0    0    1
1|   1    0    1    1    0    0    0    1    1
2|   1    1    0    1    1    0    0    0    0
3|   0    1    1    0    1    0    1    1    0
4|   1    0    1    1    0    1    1    0    0
5|   1    0    0    0    1    0    1    0    1
6|   0    0    0    1    1    1    0    1    1
7|   0    1    0    1    0    0    1    0    1
8|   1    1    0    0    0    1    1    1    0

Con una frequenza empirica di 15747 ottengo un taglio minimo
Probabilità di ottenere un taglio minimo: 0.15747
Run necessarie per ottenere almeno un taglio minimo con una probabilità del 99,9%: 41
```

Si osserva nell'output la stampa del grafo sottoforma di matrice e la stampa delle statistiche richieste dal compito.

Riguardo le statistiche si denota che con un maggior numero di iterazioni dell'algoritmo si ottiene con più frequenza un taglio minimo. Questo però non determina un aumento di probabilità: essa, infatti, si attesta attorno al 16% circa (piccola, e rimane tale anche con un maggiore o minore numero di iterazioni), in linea con il fatto che l'algoritmo restituisce un taglio minimo solo se, a ogni iterazione, non campiona mai archi che compongono il taglio minimo stesso. Inoltre si osserva che l'algoritmo restituisce un taglio minimo con probabilità > 2/n^2: infatti 0.15747 > 0.00002. Infine per il calcolo del numero di iterazioni necessarie per avere una probabilità di taglio minimo pari al 99,9% si è utilizzata la serie geometrica, per cui la probabilità di ottenere almeno un taglio in n lanci è uguale al complemento a 1 della probabilità di ottenere n volte il contrario.