Libreria "grafi.c"

Una libreria di funzioni utili per l'implementazione di algoritmi su grafi in linguaggio C

Corso di Ottimizzazione Combinatoria (IN440)
a.a. 2013-2014

Strutture per liste e code

```
1. typedef struct nodo {
2.    int info;
3.    struct nodo *next;
4. } elementoLista;
```

```
1. struct nodo {
2.    int info;
3.    struct nodo *next;
4. };
```

```
5. typedef struct scoda {
6.    elementoLista *primo;
7.    elementoLista *ultimo;
8. } coda;
```

Gestione di liste come “pile” (last in first out)

```
1. elementoLista *impila(elementoLista *p, int x) {
2.   elementoLista *q;
3.   q = malloc(sizeof(elementoLista));
4.   q->info = x;
5.   q->next = p;
6.   p = q;
7.   return(p);
8. }
```

Gestione di liste come “pile” (last in first out)

```
1. elementoLista *leggiLista(void) {
2.   int i, n, x;
3.   elementoLista *primo = NULL;
4.   printf("Numero di elementi: ");
5.   scanf("%d", &n);
6.   printf("Inserisci %d elementi: ", n);
7.   for (i=0; i<n; i++) {
8.     scanf("%d", &x);
9.     primo = impila(primo, x);
10.  }
11.  return(primo);
12. }
```

Gestione di liste come “pile” (last in first out)

```
1. void stampaLista(elementoLista *p) {
2.   while (p != NULL) {
3.     printf("%d --> ", p->info);
4.     p = p->next;
5.   }
6.   printf("NULL\n");
7.   return;
8. }
```

Gestione di liste come “pile” (last in first out)

```
1. elementoLista *invertiLista(elementoLista *p) {
2.   elementoLista *q = NULL;
3.   if (p != NULL && p->next != NULL) {
4.     q = invertiLista(p->next);
5.     p->next->next = p;
6.     p->next = NULL;
7.   } else {
8.     q = p;
9.   }
10.  return(q);
11. }
```

Gestione di una coda

```
1.  void push(coda *pQ, int x) {
2.    elementoLista *p;
3.    p = malloc(sizeof(elementoLista));
4.    p->info = x;
5.    p->next = NULL;
6.    if (pQ->ultimo == NULL){
7.      pQ->ultimo = p;
8.      pQ->primo = p;
9.    }  else {
10.     pQ->ultimo->next = p;
11.     pQ->ultimo = p;
12.   }
13.   return;
14. }
```

```
1.  int main(void) {
2.    coda Q;
3.    push(&Q, 17);
4.    push(&Q, 345);
5.    ...
6.  }
```

Gestione di una coda

```
1.  int pop(coda *pQ) {
2.    int x;
3.    elementoLista *p;
4.    x = pQ->primo->info;
5.    if (pQ->primo->next == NULL) {
6.      free(pQ->primo);
7.      pQ->primo = NULL;
8.      pQ->ultimo = NULL;
9.    } else {
10.     p = pQ->primo;
11.     pQ->primo = pQ->primo->next;
12.     free(p);
13.   }
14.   return(x);
15. }
```

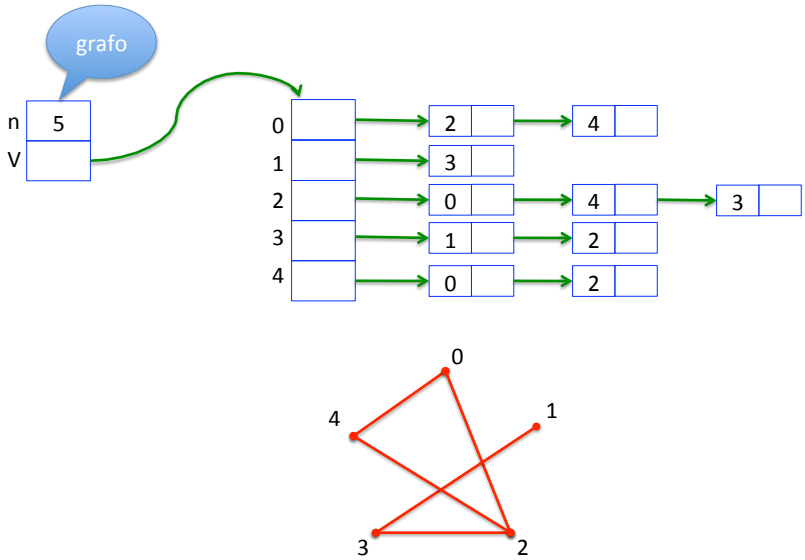
```
1.  int main(void) {
2.    coda Q;
3.    int a;
4.    ...
5.    while (Q.primo != NULL) {
6.      a = pop(&Q);
7.      printf("%d", a);
8.    }
9.    ...
10. }
```

Struttura per grafi e grafi pesati

```
1. typedef struct sgrafo {
2.   int n;
3.   elementoLista **V;
4. } grafo;


5. typedef struct sgrafoPesato {
6.   int n;
7.   elementoLista **V;
8.   int **W;
9. } grafoPesato;
```

Gestione di grafi



Gestione di grafi

```
1.  int leggiGrafo(grafo *pG) {
2.    int i, n;
3.    printf("Numero di vertici: ");
4.    scanf("%d", &n);
5.    pG->n = n;
6.    pG->V = malloc((n+1)*sizeof(elementoLista *));
7.    for (i=1; i<=n; i++) {
8.      printf("Lista di adiacenza del vertice %d.\n", i);
9.      pG->V[i] = leggiLista();
10.   }
11.   return(n);
12. }
```

Gestione di grafi

```
1.  void stampaGrafo(grafo G) {
2.    int i;
3.    for (i=1; i<=G.n; i++) {
4.      printf("%3d: ", i);
5.      stampaLista(G.V[i]);
6.    }
7.    return;
8.  }
```

Gestione di grafi

```
1.  void salvaGrafo(grafo G) {
2.    int i;
3.    FILE *out;
4.    elementoLista *p;
5.    char nome[100];
6.    printf("Nome del file: ");
7.    scanf("%s", nome);
8.    out = fopen(nome, "w+t");
9.    if (out != NULL) {
10.     for (i=1; i<=G.n; i++) {
11.       p = G.V[i];
12.       while (p != NULL) {
13.         fprintf(out, "%d ", p->info);
14.         p = p->next;
15.       }
16.       fprintf(out, "\n");
17.     }
18.     fclose(out);
19.   } else {
20.     fprintf(stderr, "ERRORE: impossibile scrivere sul file '%s'.\n\n", nome);
21.   }
22.   return;
23. }
```

Un esempio elementare

```
1.  #include <stdlib.h>
2.  #include <stdio.h>
3.  #include "grafi.h"

4.  int main(void) {
5.    grafo G;
6.    int n;
7.
8.    n = leggiGrafo(&G);
9.    salvaGrafo(G);
10.   stampaGrafo(G);
11.   return(0);
12. }
```

Un esempio elementare

Compilazione della libreria:

```
gcc -c grafi.c –Wall
```

Compilazione del programma con la libreria:

```
gcc esempio.c grafi.o -o esempio –Wall
```

Esecuzione del programma:

```
./esempio
```

Gestione di grafi

```
1.  int addEdge(grafo *pG, int u, int v) {
2.    int rc = 0;
3.    elementoLista *p;
4.
5.    if (u > pG->n)
6.      rc = 1;
7.    if (v > pG->n)
8.      rc = 2;
9.    if (rc == 0) {
10.     p = pG->V[u];
11.     while (p != NULL && rc == 0) {
12.       if (p->info == v)
13.         rc = 3;
14.       p = p->next;
15.     }
16.     if (rc == 0) {
17.       p = malloc(sizeof(elementoLista));
18.       p->info = v;
19.       p->next = pG->V[u];
20.       pG->V[u] = p;
21.     }
22.   }
23.   return(rc);
24. }
```

Gestione di grafi

```
1.  int removeEdge(grafo *pG, int u, int v) {
2.    int rc = 0;
3.    elementoLista *p, *q;
4.
5.    if (u > pG->n || v > pG->n || pG->V[u] == NULL) {
6.      rc = 1;
7.    } else {
8.      p = pG->V[u];
9.      if (p->info == v) {
10.       pG->V[u] = p->next;
11.     } else {
12.       q = p;
13.       p = p->next;
14.       while (p != NULL) {
15.         if (p->info == v) {
16.           q->next = p->next;
17.           p = NULL;
18.         } else {
19.           q = p;
20.           p = p->next;
21.         }
22.       }
23.     }
24.   }
25.   return(rc);
26. }
```

Costruzione di grafi

```
1.  int completeGraph(grafo *pG, int n) {
2.    int i, j;

3.    pG->n = n;
4.    pG->V = malloc((n+1)*sizeof(elementoLista *));
5.    for (i=1; i<=n; i++) {
6.      pG->V[i] = NULL;
7.      for (j=1; j<=n; j++) {
8.        if (i != j)
9.          addEdge(pG, i, j);
10.     }
11.   }
12.   return(n);
13. }
```

Costruzione di grafi

```
1.  int emptyGraph(grafo *pG, int n) {
2.    int i;

3.    pG->n = n;
4.    pG->V = malloc((n+1)*sizeof(elementoLista *));
5.    for (i=1; i<=n; i++)
6.      pG->V[i] = NULL;
7.    return(n);
8.  }
```

Costruzione di grafi

```
1.  int cycleGraph(grafo *pG, int n) {
2.    int i;

3.    pG->n = n;
4.    pG->V = malloc((n+1)*sizeof(elementoLista *));
5.    for (i=1; i<=n; i++)
6.      pG->V[i] = NULL;
7.    for (i=1; i<n; i++) {
8.      addEdge(pg, i, i+1);
9.      addEdge(pg, i+1, i);
10.   }
11.   addEdge(pg, n, 1);
12.   addEdge(pg, 1, n);
13.   return(n);
14. }
```

Costruzione di grafi

```
1.  int randomGraph(grafo *pG, int n, float p) {
2.    int i, j;

3.    srand((unsigned) time(NULL));
4.    pG->n = n;
5.    pG->V = malloc((n+1)*sizeof(elementoLista *));
6.    for (i=1; i<=n; i++)
7.      pG->V[i] = NULL;
8.    for (i=1; i<n; i++) {
9.      for (j=i+1; j<=n; j++) {
10.       if ((float)(rand() % 100)/100.0 < p) {
11.         addEdge(pG, i, j);
12.         addEdge(pG, j, i);
13.       }
14.     }
15.   }
16.   return(n);
17. }
```

Costruzione di grafi

```
1.  void grafoTrasposto(grafo G, grafo *pGT) {
2.    int i;
3.    elementoLista *p, *q;
4.    pGT->n = G.n;
5.    pGT->V = malloc((G.n+1)*sizeof(elementoLista *));
6.    for (i=1; i<=G.n; i++) {
7.      pGT->V[i] = NULL;
8.    }
9.    for (i=1; i<=G.n; i++) {
10.     p = G.V[i];
11.     while (p != NULL) {
12.       q = malloc(sizeof(elementoLista));
13.       q->info = i;
14.       q->next = pGT->V[p->info];
15.       pGT->V[p->info] = q;
16.       p = p->next;
17.     }
18.   }
19.   return;
20. }
```

Esercizio

Dato un grafo random G=(V,E)
costruire il grafo complementare GC=(V, E^C)