Prima soluzione (sol_quad.c)

```
// Coppia contenente k-gramma e conteggio delle sue
//occorrenze

struct Pair{
 char *kgram;   // puntatore al k-gramma
 int occ;       // numero occorrenze
};

int K;          //lunghezza dei k-grammi


// funzione confronto lessicografico tra k-grammi
int comp_kgrams(char *a, char *b){
 int i;
 for (i=0; i< (K-1) && (a[i] == b[i]); i++);
 return (a[i]-b[i]);
}
```

Lettura input:

```
char s[1000];
struct Pair* kfreq;
int T;
int len, i, j,distinct;

// lettura dei parametri in input
scanf("%d",&K);
scanf("%d",&T);
scanf("%s",s);
len = strlen(s);

if (len < K) return;
```

Conteggio in tempo quadratico

```
kfreq = malloc((len-K+1) *sizeof(struct Pair));

for (i=0; i<= len-K; i++) {
  // determina se è la prima occorrenza...
  for (j=0; j<i; j++)
   if (comp_kgrams(s+i, s+j) == 0) break;

  // ... in caso affermativo:
  if (j == i)  {
   kfreq[distinct].kgram = s+i;
   kfreq[distinct].occ = 0;

   for (j=0; j <= len-K; j++)
    if (comp_kgrams(s+i, s+j) == 0)
          kfreq[distinct].occ++;
   distinct ++;
  }
```

Ordinamento per frequenze

```
// ordina per occ decrescente
qsort(kfreq,distinct,sizeof(struct Pair),comp_pair);

for (i=0; i< T && i < distinct; i++){

     // stampa del k-gramma
     fwrite(kfreq[i].kgram, sizeof(char), K, stdout);
     printf(" %d\n", kfreq[i].occ);


 }
```

frequenze

```
int comp_pair(const void *_a, const void *_b) {

  struct Pair *a = (struct Pair*)_a;
  struct Pair *b = (struct Pair*)_b;

  if (a->occ != b->occ)
     return (b->occ - a->occ);
  else
     return comp_kgrams(a->kgram, b->kgram);

}
```

```
                                    Pair),comp_pair);

for (i=0; i< r && i < distinct; i++){

        // stampa del k-gramma
        fwrite(kfreq[i].kgram, sizeof(char), K, stdout);
        printf(" %d\n", kfreq[i].occ);


  }
```

Conteggio con sorting

E' possibile abbassare la complessità del conteggio da *$O(n^2)$ ad $O(n \log n)$ confronti tra k-grammi* utilizzando *algoritmi di ordinamento:*

```
 // array di tutti i k-grammi di s
char** kgrams = malloc(sizeof(char*)*(len-K+1));

 for (i=0; i<= (len-K); i++)
  kgrams[i] = s+i;

 qsort(kgrams, len-K+1, sizeof(char *),
comp_kgrams_void);
```

Conteggio con sorting

Si scandisce l'array ordinato *kgram* e si conta il numero di volte che ciascun k-gramma appare di fila:

```
for (i=1; i <= (len-K); i++) {
   if (comp_kgrams(kgrams[i],
                   kgrams[i-1]) == 0)
      kfreq[distinct].occ++;

   else{
      distinct++;
      kfreq[distinct].kgram = kgrams[i];
      kfreq[distinct].occ = 1;
    }
}
```

Esercizio

Scrivere un programma che presa in input una stringa s e due interi k,t determina se nella stringa esiste un k-gramma che occorre più di t volte e in caso affermativo lo stampa.

NOTA: La soluzione deve indicizzare i k-grammi in una tabella hash di dimensione 2*N dove N = |s| sfruttando la tecnica di hashing di Rabin

```
// entry della lista di trabocco della tabella
struct Nodo{
   char *kgram;
   int occ;
   struct Nodo* succ;
};
```

Hash alla Karp-Rabin:

```
#define P (999149)
// costruzione vettore potenze del due mod P
 pow[0] = 1;
 for (i=1; i<K; i++)
   pow[i] = (pow[i-1]*2)%P;

 // inizializza valore di hash
 hash=0;
 for (i=0; i<K; i++)
   hash = (hash + s[i]*pow[K-1-i]) %P;

 // scorre la stringa e aggiorna di volta in volta hash
 for (i=0; i<= len-K; i++) {
  //  .... inserzione nella tabella in posizione (hash mod 2*N)

  hash = (hash + P - (s[i]*pow[K-1])%P ) %P;
  hash = (2*hash + s[i+K]) % P;
```

Hash alla Karp-Rabin:

```
#define P (999149)
// costruzione vettore potenze del due mod P
 pow[0] = 1;
 for (i=1; i<K; i++)
   pow[i] = (pow[i-1]*2)%

 // inizializza valore di ha
 hash=0;
 for (i=0; i<K; i++)
   hash = (hash + s[i]*pow

 // scorre la stringa e aggiorna    olta in volta hash
 for (i=0; i<= len-K; i++) {
  //  .... inserzione nella tabella in posizione (hash mod 2*N)

  hash = (hash + P - (s[i]*pow[K-1])%P ) %P;
  hash = (2*hash + s[i+K]) % P;
```

Verifica se il k-gramma i-esimo è presente nella lista di trabocco, se si incrementa il valore di *occ*, altrimenti inserisce un nuovo nodo nella lista