Laboratorio di Fondamenti di Informatica 3/12/2015

Anno accademico 2015/2016

Esercizio 0 – Ricerca di Parole nelle Stringhe
Un esercizio facile, ma che ci servirà in futuro

[Questo è un pre-esercizio, serve ad avere un utile strumento per un esercizio futuro. Non c'entra né con la ricorsione né con la memoria dinamica]

Si considerino due stringhe: una *frase*, che può contenere spazi e separatori, ed è in generale costituita da più parole e da caratteri non alfabetici, e una *parola*, che contiene solo caratteri alfabetici.

Si codifichi la funzione "parola-in-frase" (p-i-f):

```
int pif( char * frase, char * parola )
```

che restituisce 1 se la *parola* è contenuta nella *frase*, 0 altrimenti.

Possibilmente, il riconoscimento sia case-insensitive.

*Attenzione: si vuole realizzare un comportamento diverso da quello della funzione strstr(). La parola deve comparire come parola "intera" a sé stante, non come sottostringa generica. Tuttavia, una **brutale** approssimazione del comportamento desiderato si può ottenere con:*

```
int pif( char * frase, char * parola ) {
  return strstr(frase, parola) != NULL ;
}
```

1. La potenza è nulla, senza controllo

- Si codifichi una funzione ricorsiva

```
int is_power( int n, int b )
```

  che controlla **se** un numero intero n è una potenza del numero intero b, restituendo 0 o 1.
- Per farlo, si consideri la seguente proprietà: n è una potenza di b se è divisibile per b e inoltre n/b è a sua volta una potenza di b.
- Si badi ad operare correttamente anche nei casi b=0, b=1.

2a. Rimozione di caratteri

Si scriva una funzione che, ricevendo come parametri una stringa s ed un carattere c, restituisca UNA NUOVA stringa di lunghezza minima (allocata dinamicamente) uguale a s ma priva di tutte le occorrenze del carattere c.

Esempio: passando `"esercizio semplice"` e `'s'` la funzione dovrà allocare e restituire la stringa `"eercizio emplice"`

Attenzione:

- La funzione non conosce a priori la lunghezza della stringa s.
- Si richiede esplicitamente di restituire una nuova stringa di lunghezza minima, cioè che occupi la minima quantità di memoria: occorre calcolare tale quantità prima della malloc!

2b. Concatenazione parsimoniosa

Scrivere una funzione **`conc_pars(…)`** che riceva come parametri **due stringhe**, e restituisca come risultato una nuova stringa, ottenuta dalla **concatenazione** delle stringhe ricevute, facendo in modo che la nuova stringa occupi la quantità minima di memoria.

La funzione non deve modificare le stringhe ricevute. Deve invece restituire un(puntatore all)a nuova stringa, che deve essere **allocata dinamicamente** (o non potrebbe essere utilmente restituita dalla funzione!).

Esempio: s1 = “Prima” s2 = "Seconda" -> restituisce = “PrimaSeconda”

2c. Moltiplicazione di stringhe

Si scriva poi una funzione **`strmul(…)`** che, dati una stringa s (*che dev'essere stata allocata dinamicamente*) e un numero intero n, **modifichi** la stringa s, riallocandola e assegnandole il risultato della concatenazione di s con se stessa per n volte. Si noti che

- s **deve** essere stata allocata dinamicamente (o la funzione non potrebbe deallocarla e sostituirla con una più lunga!)
- s deve essere opportunamente riallocata in modo da contenere esattamente il risultato della "moltiplicazione" (dimensione minima)
- per essere modificata dalla funzione, la stringa s (che nell'ambiente del programma chiamante è un puntatore a carattere) deve essere passata per indirizzo.

Esempio: s = "abC" -> moltiplicazione per 3 -> s = "abCabCabC"

Suggerimento: si possono usare le funzioni strcat() e/o conc_pars() ?

2d. Raddoppio di caratteri in un vettore di parole

Si scriva una funzione che, ricevendo come parametri:

- un array *sorgente* (di dimensione nota) di puntatori (che puntano a stringhe senza spazi),
- un secondo array *destinazione* (della stessa dimensione) di puntatori a stringhe, tutti a NULL,
- un carattere che rappresenta la lettera da raddoppiare

per ogni stringa del primo array allochi dinamicamente una nuova stringa, di dimensione minima, ottenuta raddoppiando la consonante passata se parametro della prima stringa; la funzione dovrà popolare correttamente l'array di destinazione.

Esempio: passando come parametro la lettera c un siffatto array sorgente, l'array di destinazione dovrà diventare:

Sorgente	Destinazione
Casa	Ccasa
Scuola	Sccuola
Macchina	Macccchina
Bicicletta	Biccicceletta

3. Media Mobile Variabile

Scrivere un programma che, data una sequenza arbitraria di numeri in virgola mobile immessi uno alla volta dall'utente, stampi man mano la media degli ultimi n valori immessi, dove anche n viene immesso dall'utente all'avvio del programma. In ogni momento l'utente può scegliere se a) aggiungere un valore alla sequenza, b) modificare il valore di n, oppure c) terminare il programma.

I numeri sono immessi fino all'immissione del valore "sentinella" 0 che termina il programma. Ogni volta che è immesso un numero il programma mostra la media mobile aggiornata. Quando, all'inizio, sono stati inseriti meno di n numeri, o quando n viene aumentato, la media viene comunque calcolata sui valori disponibili. Si badi a tratare correttamente anche il caso in cui n viene diminuito.

Suggerimenti:

- Si usi come "buffer circolare" un vettore di n elementi allocato dinamicamente per memorizzare gli ultimi n valori inseriti: i valori sono aggiunti uno dopo l'altro, ripartendo dall'inizio quando si raggiunge la fine dell'array.
- **Si scomponga il problema in sottofunzioni**

4. FloodFill (coloriamo il mondo)

Capita spesso di dover “colorare” o generalmente marcare delle aree continue ma irregolari, delimitate da un bordo o dalla contiguità con aree che hanno proprietà diverse. Si usa, in questi casi, un tipico algoritmo che, a partire da un punto iniziale, procede “a macchia d’olio”. Si consideri una mappa inizialmente “a due colori” (pieno e vuoto) e si implementino le seguenti funzioni:

1. Partendo da un punto “pieno”, colorare uniformemente con un colore P tutta l’area “piena” a cui appartiene il punto
2. Analogamente, da un punto “vuoto”, colorare uniformemente con un colore V la corrispondente area vuota
3. Colorare con i colori P e V l’intera mappa (e non solo l’area contigua a un punto indicato)

4. FloodFill (coloriamo il mondo)

Attenzione:

Il problema ammette una elegante soluzione ricorsiva in quanto ogni punto della mappa può essere colorato e la funzione di colorazione può essere invocata su tutti i punti ad esso adiacenti che ancora non siano stati colorati.

4. FloodFill (coloriamo il mondo)

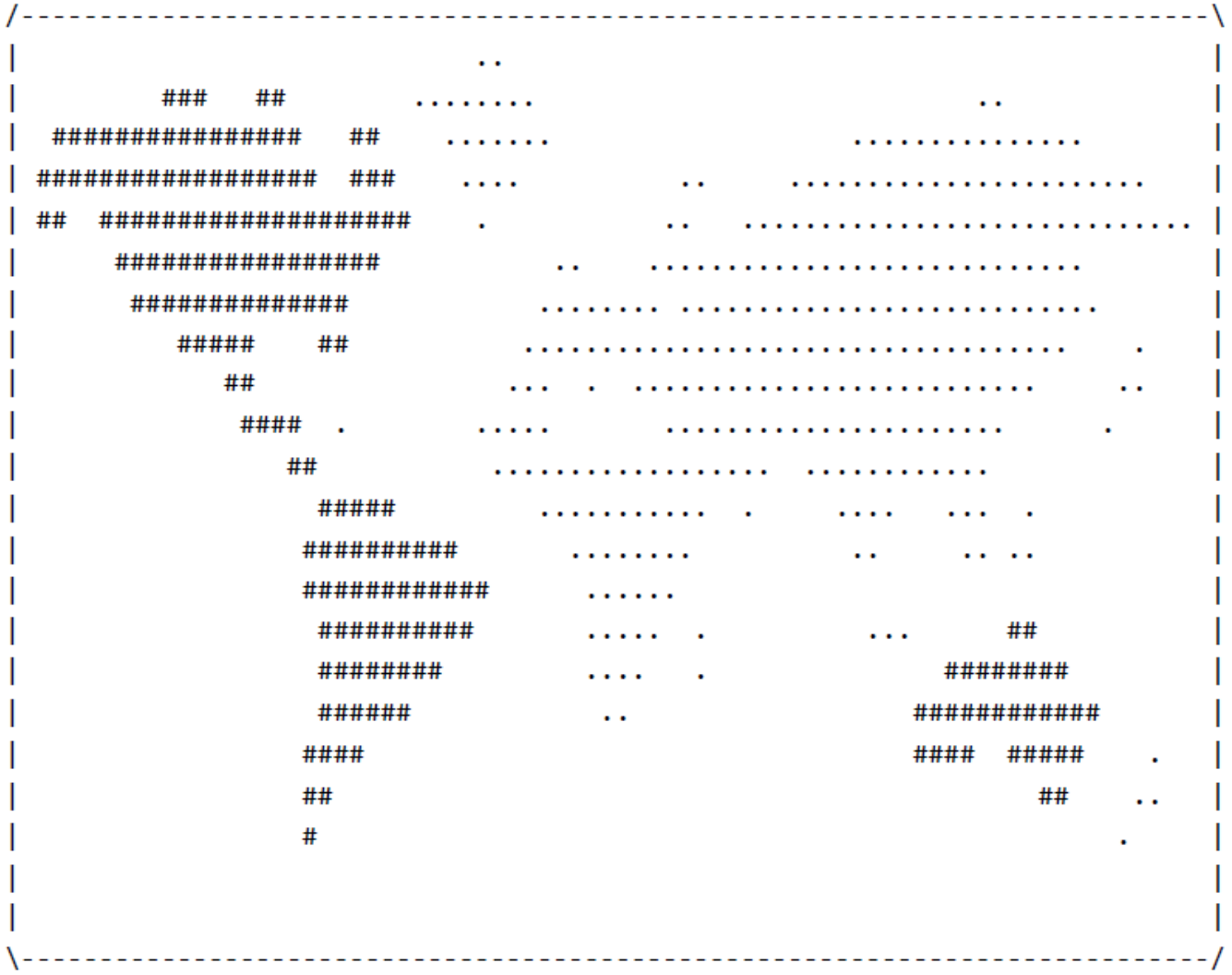
4. Richiedere interattivamente all'utente le coordinate di un punto e
   - Se si "cade" su un'area piena e non colorata, colorarla col colore P
   - Se si "cade" su un'area vuota e non colorata, colorarla col colore V
   - Se si "cade" su un'area già colorata, riportarla allo stato iniziale (togliere il colore P/V e rimettere)
5. (*più difficile*) Colorare ogni area piena con un colore *diverso* da tutte le altre aree piene disgiunte, indicando al termine della funzione il numero di aree colorate (cioè il numero di colori diversi utilizzati)

In particolare, si considerino per fissare le idee delle mappe rettangolari a caratteri in cui inizialmente il pieno e il vuoto sono rispettivamente rappresentati dai caratteri ' . ' e ' ', da interpretarsi "topograficamente" come terra e acqua, e si usino come "colori" i caratteri '#' per colorare le isole e '~' per colorare i mari. Per l'ultimo punto si possono usare i "colori" '1', '2', ... '9'

4. FloodFill (coloriamo il mondo)

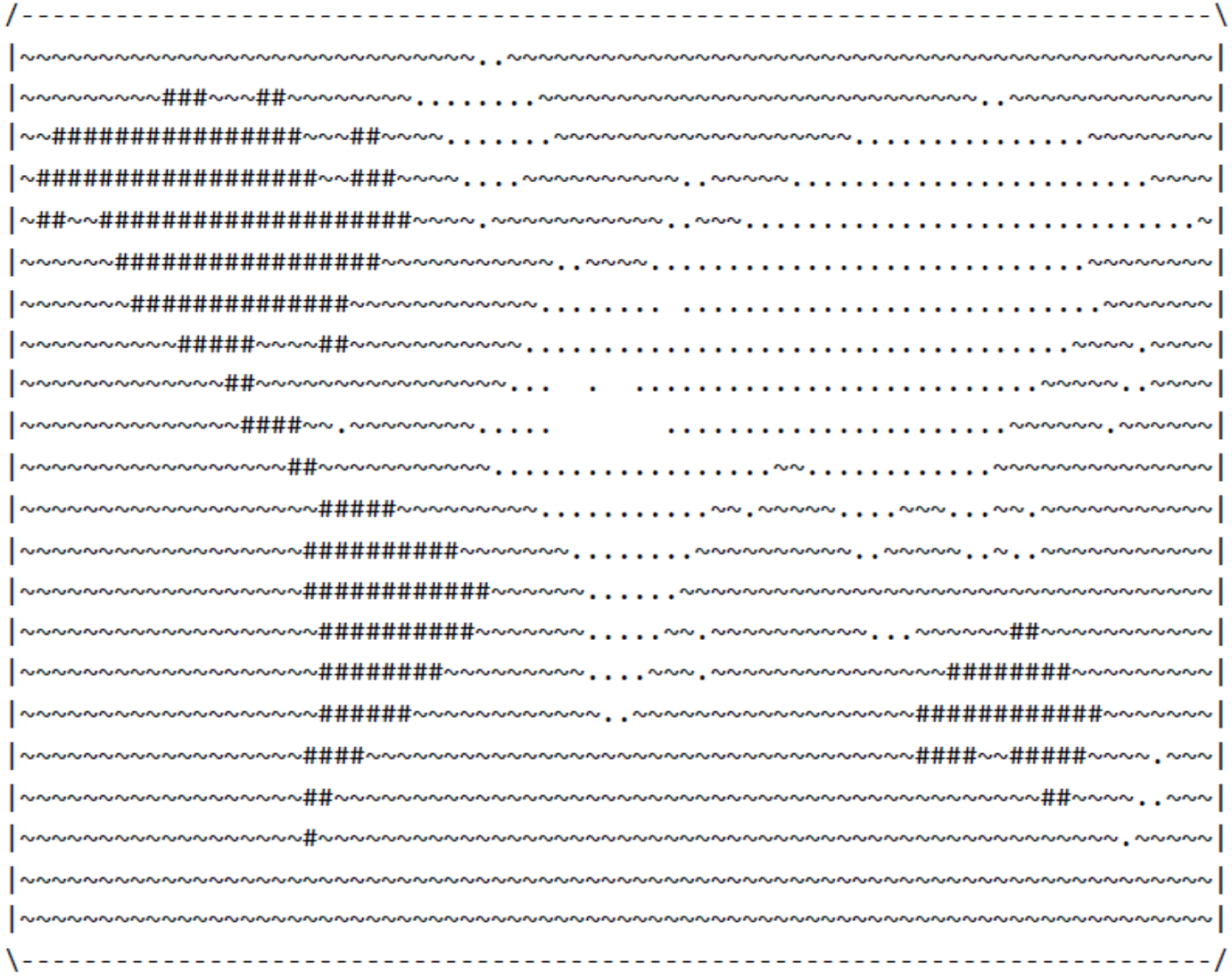
Mappa iniziale (pieni e vuoti non colorati)

4. FloodFill (coloriamo il mondo)



Due isole di “terra” sono state colorate

4. FloodFill (coloriamo il mondo)



Anche il mare è stato colorato

5. Boggle

- Il *Boggle* (il *Paroliere*, nell'edizione italiana) è un gioco di abilità che consiste nel comporre le parole più lunghe possibili con le lettere di uno schema NxN, iniziando da una casella arbitraria e "serpeggiando" in ogni possibile direzione. Ad esempio, nello schema a margine leggiamo META, SUINO, STRINU', TIRANTE, SPARTANI ...

E	**N**	**A**	**P**
M	**T**	**S**	**R**
D	**U**	**I**	**T**
C	**O**	**N**	**A**

- Se si ammette che una lettera possa essere usata più di una volta (versione che chiamiamo ***easy***) possiamo leggere anche parole come MENT**E**, MUT**U**O, AN**ANA**S, TEN**ENTE**, CONTI**N**U**IT**A' e CO**CCO** (si noti che la doppia C è ottenuta riusando la lettera C due volte di fila "senza spostarsi").
- Se invece si aggiunge il vincolo di non riusare le lettere (versione ***classica***), le parole ammissibili sono di meno, e sono ovviamente limitate in lunghezza.

5. Boggle

Consideriamo dapprima la versione ***easy*** e verifichiamo se una data parola è ammissibile o no per un dato schema. Si codifichi una funzione **int** verificaEasy(**char** p[], **char** s[][N]), che restituisce **1** se p è leggibile nello schema s, **0** altrimenti

Potenziali "aiutini" :

- *può essere utile definire funzioni di servizio, ad esempio per verificare una parola (o una sua parte) a partire da una data cella;*
- *si badi a non "uscire" dallo schema con accessi invalidi alla memoria;*
- ***può** risultare semplice/comodo ragionare ricorsivamente:*

  ***casi base**:*

  *- se p è vuota, la parola è banalmente **leggibile** in s*
  *- se i o j sono fuori dai limiti di s (<0 oppure >= N), p ovviamente **non** è **leggibile** in s da i,j*
  *- se il carattere s[i][j] è diverso dal carattere p[0], p ovviamente **non** è **leggibile** in s da i,j*

  ***passo induttivo**:*

  assumiamo quindi di saper verificare se una parola più corta di p (in particolare, la coda di p, cioè la parola che inizia da p+1) è leggibile da una data cella di s. Allora se (a) il primo carattere di p è proprio il carattere s[i][j] e inoltre (b) a partire da una almeno delle celle adiacenti alla cella i,j (o dalla cella i,j stessa) possiamo leggere la coda di p, allora anche l'intera p è leggibile in s. Se la coda non è leggibile da nessuna cella adiacente, allora p non è leggibile a partire dalla cella i,j

Naturale estensione è il passaggio alla versione ***classica*** (ogni lettera è usata una volta sola)

5+. Boggle (challenge version)

Con riferimento alla versione ***classica*** del gioco (in cui ogni lettera è usata una e una sola volta) si codifichi la seguente funzione:

```
void stampatutte( …, int n )
```

che stampa (a video, o - in futuro - in un file) tutte le parole di lunghezza n (naturalmente compreso tra 1 e NxN) che si possono formare con un dato schema.

Ma… come si fa a stampare solo le parole «di senso compiuto» ??