INFORMATICA 1
ESERCITAZIONI
Corso di Laurea in Fisica
a.a. 2010/11

Andrea Corradini, Roberta Gori

[1]Dipartimento di Informatica

Schemi di programma: ricerca e verifica

- Molti problemi riguardano la ricerca di elementi in intervalli o la verifica di proprietà.
- Sviluppiamo schemi di programma (dimostrabilmente corretti) che realizzano la ricerca e la verifica.
- La soluzione di problemi concreti consiste poi nella sostituzione di alcuni parametri degli schemi con valori specifici dei problemi in esame.
- Distinguiamo due tipi di ricerca: ricerca certa e ricerca incerta.
  - **ricerca certa**: si vuole determinare il minimo elemento di un intervallo [a,b) per il quale vale una certa proprietà $\mathcal{P}$, sapendo che almeno un elemento dell'intervallo soddisfa $\mathcal{P}$.
  - **ricerca incerta**: si vuole determinare, se esiste, il minimo elemento di un intervallo [a,b) per il quale vale una certa proprietà $\mathcal{P}$.

Ricerca certa

- Intervallo di ricerca: [a,b)
- Proprietà: $\mathcal{P}(\cdot)$
- Ipotesi di certezza: $\exists\, i \in [a,b)\ .\ \mathcal{P}(i)$
- Stato finale:
  $x = \min\ \{\, i \in [a,b)\ \mid\ \mathcal{P}(i)\}$.
- Lo schema generale per risolvere il problema è il seguente:

```
int x;
x=a;
while (!P(x))
   x=x+1;
```

- Nota: l'estremo destro dell'intervallo non serve.
- Si assume che la proprietà $\mathcal{P}$ sia esprimibile nel linguaggio.

Ricerca certa: esempio 1

- Calcolare la radice intera di un numero naturale.
- Si può esprimere come problema di ricerca certa:
  $\lfloor\sqrt{N}\rfloor = \min\{x \in [0,N+1) \mid x^2 \leq N < (x+1)^2\}$
- Dunque l'estremo sinistro dell'intervallo di ricerca, a nello schema, in questo caso è 0, mentre l'estremo destro, b nello schema, è N.
- Infine la proprietà $\mathcal{P}(x)$ dello schema è $N < (x+1)^2$

```
int x;
x=0;
while ((x+1)*(x+1) <= N)
  x=x+1;
```

Ricerca certa: esempio 2

- Determinare la posizione della prima occorrenza di un dato elemento in un array, sapendo che tale elemento vi occorrre almeno una volta.
- Indichiamo con vet l'array e con DIM la sua dimensione
- Vogliamo determinare:
  $x = \min\{i \in [0,DIM) \mid vet[i] = el\}$
- Possiamo istanziare lo schema come segue:

```
int x;
x=0;
while (vet[x]!=el)
  x=x+1;
```

Ricerca Incerta

- Si vuole determinare, se esiste, il minimo elemento di un intervallo [a,b) per il quale vale una certa proprietà $\mathcal{P}$.
- Perché lo schema di ricerca certa non va bene?

```
x=a;
while (!P(x))
  x=x+1;
```

- Se l'elemento non c'è si vanno ad esaminare valori di x che sono al di fuori dell'intervallo di ricerca e per i quali la la proprietà $\mathcal{P}$ potrebbe addirittura non essere definita (errore a tempo di esecuzione).

  Esempio: Nel caso della ricerca incerta di un elemento in un array di dimensione DIM si andrebbero ad esaminare elementi del tipo vet[x] con x >= DIM.
- Abbiamo bisogno di modificare lo schema in modo che l'analisi degli elementi avvenga solo all'interno dell'intervallo di ricerca e che la ricerca venga interrotta una volta esaurito l'intervallo (e non individuato alcun elemento).

Ricerca incerta

- Intervallo di ricerca: [a,b)
- Proprietà: $\mathcal{P}(\cdot)$
- Stato finale: $x = \min\{i \in [a,b) \mid \mathcal{P}(i)\}$ *min* b

  $\Longrightarrow$ dobbiamo stabilire quale valore calcolare se nessun elemento dell'intervallo soddisfa $\mathcal{P}$: una buona scelta è il valore b, che sicuramente non fa parte dell'intervallo.

Ricerca incerta

- Utilizziamo una variabile booleana `trovato` che fa da sentinella

  $\Longrightarrow$ impone l'uscita dal ciclo non appena si individua un elemento che soddisfa la proprietà
- in congiunzione con la sentinella, la guardia del ciclo assicura che l'intervallo di ricerca non sia esaurito

```
int trovato = FALSE; /* inizialmente false */
int x=a;
while (!trovato && x<b)
  if (P(x))
    trovato = TRUE; /*x soddisfa P */
  else
    x=x+1;
```

- Si suppone che le costanti `TRUE` e `FALSE` siano state definite opportunamente, ad esempio mediante le direttive

```
#define FALSE 0
#define TRUE 1
```

All'uscita dal ciclo abbiamo due possibilità:

1. trovato $\wedge$ x $\in$ [a,b) $\wedge$ x = min$\{$i$\in$ [a,b) | $\mathcal{P}$(i)$\}$
   - x è l'indice cercato
   - il valore di verità di trovato è TRUE

2. $\neg$ trovato $\wedge$ x=b $\wedge\forall$ i$\in$ [a,b). $\neg$ $\mathcal{P}$(i)
   - nessun elemento nell'intervallo di ricerca soddisfa $\mathcal{P}$
   - il valore di x è b (fuori dell'intervallo)
   - il valore di verità di trovato è FALSE

Ricerca incerta: esempio

- Determinare la prima occorrenza di un elemento in un array.
- È un problema di ricerca incerta:
  $\min \{x \in [0,DIM) \mid vet[x] = el\}$ *min DIM*

```
int trovato = FALSE;
int x=0;
while (!trovato && x<DIM)
  if (vet[x]==el)
    trovato = TRUE;
  else
    x=x+1;
```

- Vi sono situazioni in cui la proprietà $\mathcal{P}$ della ricerca (certa o incerta) non è direttamente esprimibile nel linguaggio.
  Esempio: Determinare (se c'è) la posizione del primo elemento di un array di interi che è uguale alla somma degli elementi che lo precedono.
- Si tratta di un problema di ricerca incerta in cui
  1. l'intervallo [a,b] è [0,DIM]
  2. la proprietà $\mathcal{P}(x)$ è

$$vet[x] = \sum_{j=0}^{x-1} vet[j]$$

```
int trovato = FALSE;
int x=0;
while (!trovato && x<DIM)
    if (vet[x]== Σ_{j=0}^{x-1} vet[j])
        trovato = TRUE;
    else
        x=x+1;
```

- In questi casi si utilizza la seguente tecnica:
  1. si rimpiazzano le espressioni critiche con variabili
  2. si impone l'uguaglianza tra le variabili cosi' introdotte e le corrispondenti espressioni "critiche", aggiungendo quanto necessario al corpo del ciclo per mantenere vere tali uguaglianze
- Nell'esempio:

```
int trovato = FALSE;
int x=0;
int sommaPrecedenti = 0;
while (!trovato && x<DIM)
  if (vet[x]==sommaPrecedenti)
    trovato = TRUE;
  else
    { sommaPrecedenti = sommaPrecedenti + vet[x];
      x=x+1;     }
```

Verifica di una proprietà

- Vogliamo verificare che tutti gli elementi di un intervallo soddisfano una certa proprietà $\mathcal{P}$.
  1. Facciamo una ricerca incerta del minimo elemento dell'intervallo per il quale non vale la proprietà $\mathcal{P}$
  2. Se non troviamo tale minimo, la verifica ha esito positivo, altrimenti ha esito negativo.
- Lo schema generale per risolvere questo problema.

```
int trovato = FALSE;
int x=a;
while (!trovato && x<b)
  if (!P(x))
    trovato = TRUE;
  else
    x=x+1;
if (trovato)
  /* esito negativo */
else
  /* esito positivo */
```

Esempio: Sono dispari tutti gli elementi di un array? Cerchiamo il primo elemento pari!

```
{
  int trovato=FALSE;
  int i=0;
  while (!trovato && i<dim)
     if (vet[i] % 2 == 0)
          trovato=TRUE;
    else
          i=i+1;

 if (!trovato)
   printf(''Gli elementi sono tutti dispari'');
 else
  printf(''Gli elementi non  sono tutti dispari'');
}
```