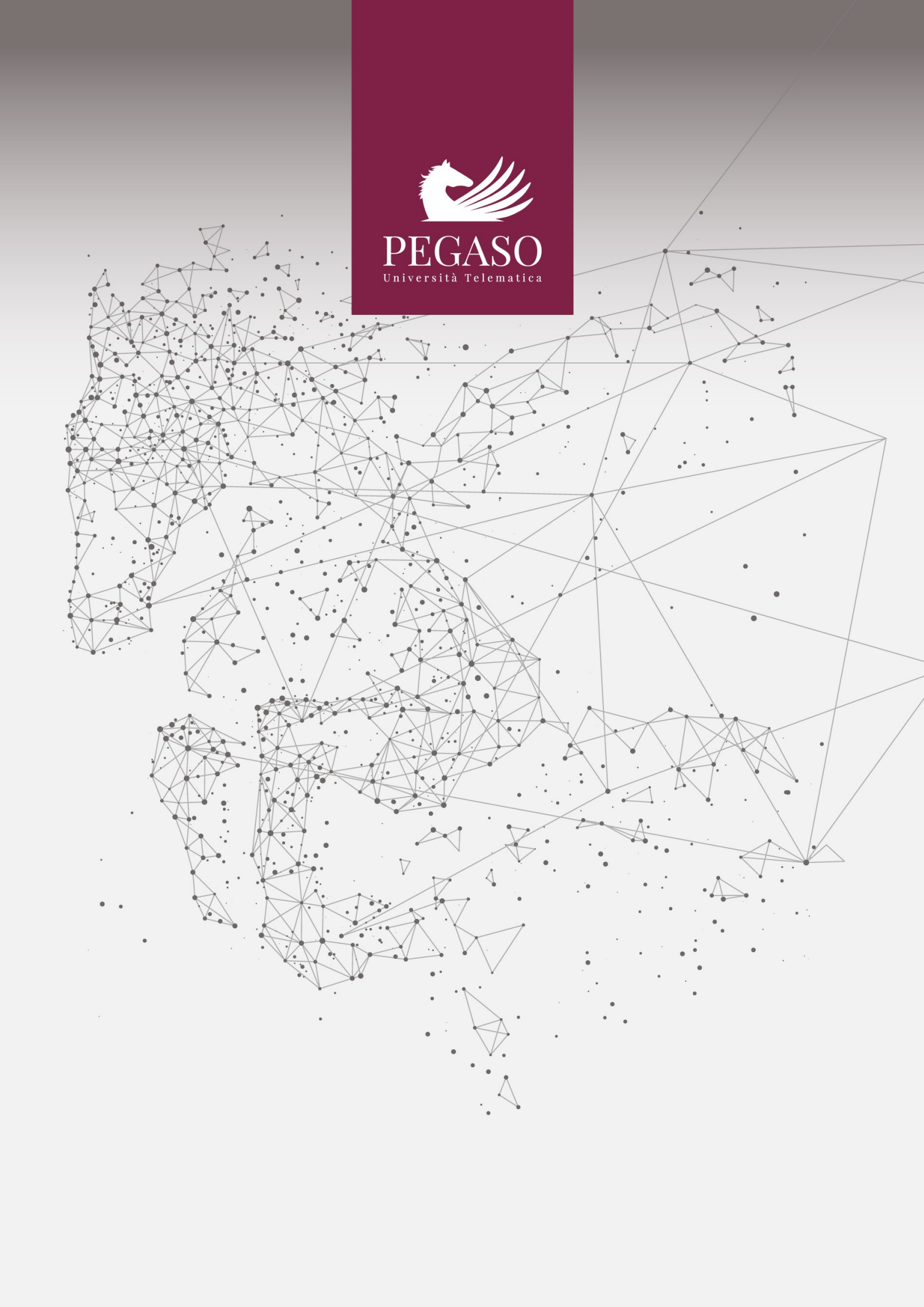
PEGASO
Università Telematica

Indice

1. Algoritmo di Las Vegas

Nel campo della programmazione, esistono numerose strategie utilizzate per risolvere problemi e ottenere risultati in modo efficiente. Tra queste strategie, si distinguono gli **algoritmi deterministici** e gli **algoritmi probabilistici**.

Gli algoritmi deterministici sono basati su una serie di passi logici che producono una soluzione esatta e deterministica al problema in questione. Questi algoritmi sono utilizzati quando la soluzione esatta del problema può essere ottenuta con certezza e senza ambiguità. Gli algoritmi deterministici sono spesso utilizzati per risolvere problemi di ottimizzazione, ricerca e ordinamento.

D'altra parte, gli algoritmi probabilistici producono soluzioni approssimate al problema in questione, basandosi sulla probabilità invece che sulla certezza. Questi algoritmi sono utilizzati quando la soluzione esatta del problema è difficile o impossibile da ottenere con certezza, ma una soluzione approssimata può essere sufficiente per le esigenze dell'applicazione. Gli algoritmi probabilistici sono spesso utilizzati per la simulazione di fenomeni fisici, la ricerca di informazioni su Internet e la criptografia.

Gli algoritmi probabilistici sono algoritmi che utilizzano la probabilità come base per prendere decisioni. Questo tipo di algoritmo è molto utilizzato in diversi campi, come la teoria dei giochi, la crittografia, l'intelligenza artificiale e molti altri.

In generale, un algoritmo probabilistico è composto da due parti:

- la fase di preparazione
- la fase di esecuzione

Nella fase di preparazione, l'algoritmo prepara i dati necessari per l'esecuzione dell'algoritmo stesso; nella fase di esecuzione, l'algoritmo utilizza i dati preparati nella fase precedente per prendere una decisione basata sulla probabilità.

L'algoritmo di Las Vegas è un algoritmo probabilistico che fornisce sempre una risposta corretta, ma può richiedere un tempo di esecuzione variabile a causa della sua natura probabilistica; in particolare, viene eseguito per un tempo casuale, ma garantisce che la risposta restituita sia corretta.

Tale algoritmo viene spesso utilizzato per risolvere problemi complessi che richiedono una grande quantità di risorse computazionali per essere risolti in modo deterministico: in questi casi può fornire una soluzione corretta in meno tempo rispetto all'implementazione deterministica.

Ecco alcuni esempi di algoritmi di Las Vegas:

- **Algoritmo di ricerca di un elemento in una tabella hash**: questo algoritmo consiste nel cercare un elemento all'interno di una tabella hash utilizzando una funzione di hash casuale. Il tempo di esecuzione dipende dalla distribuzione dei valori nella tabella hash e dalla funzione di hash utilizzata, ma la soluzione restituita è sempre corretta.
- **Algoritmo di ricerca di un elemento in un albero binario di ricerca bilanciato**: questo algoritmo consiste nel cercare un elemento all'interno di un albero binario di ricerca bilanciato utilizzando una strategia casuale per scegliere il percorso di ricerca. Il tempo di esecuzione dipende dalla struttura dell'albero e dalla strategia di ricerca utilizzata, ma la soluzione restituita è sempre corretta.
- **Algoritmo di clustering k-means probabilistico**: questo algoritmo è una variante probabilistica dell'algoritmo k-means utilizzato per il clustering di dati. In questo caso, i centroidi dei cluster vengono scelti casualmente, il che rende l'algoritmo probabilistico. Tuttavia, l'algoritmo garantisce sempre una soluzione corretta e può fornire prestazioni migliori rispetto all'algoritmo k-means deterministico in alcuni casi particolari.
- **Algoritmo di selezione del mediano**: questo algoritmo consiste nel trovare il mediano di un insieme di numeri utilizzando una strategia casuale per scegliere l'elemento di pivot. Il tempo di esecuzione dipende dalla distribuzione dei valori nell'insieme di numeri e dalla strategia di scelta dell'elemento di pivot utilizzata, ma la soluzione restituita è sempre corretta.
- **Algoritmo di Dijkstra probabilistico**: questo algoritmo viene utilizzato per trovare il percorso più breve in un grafo pesato. In questo caso, la scelta del prossimo nodo da esaminare viene effettuata casualmente, il che rende l'algoritmo probabilistico. Tuttavia, l'algoritmo garantisce sempre una soluzione corretta e può fornire prestazioni migliori rispetto all'algoritmo di Dijkstra deterministico in alcuni casi particolari.
- **Algoritmo di quicksort probabilistico**: in questo algoritmo, l'elemento di pivot viene scelto casualmente, invece di essere scelto come il primo, l'ultimo o l'elemento mediano dell'array. Questo rende l'algoritmo di quicksort probabilistico, perché il tempo di esecuzione dipende dalla scelta casuale dell'elemento di pivot. Tuttavia, l'algoritmo di quicksort probabilistico garantisce sempre una soluzione corretta.
- **Algoritmo di Rabin-Karp per la ricerca di sottostringhe in una stringa**: l'algoritmo di Rabin-Karp utilizza una funzione di hash per confrontare le sottostringhe con la stringa di ricerca, ma la scelta della funzione di hash e del valore iniziale della funzione di hash sono casuali. Ciò rende l'algoritmo di Rabin-Karp probabilistico, ma garantisce sempre una soluzione corretta.

L'algoritmo di Las Vegas è dunque utile in situazioni in cui è accettabile avere un tempo di esecuzione variabile, purché la soluzione restituita sia corretta. Tuttavia, poiché tale algoritmo è probabilistico, non è adatto a tutti i problemi e potrebbe richiedere un'analisi più approfondita per garantire che la soluzione restituita sia sempre corretta e viene utilizzato principalmente in situazioni in cui il tempo di esecuzione può variare notevolmente, ma dove è comunque importante ottenere una soluzione corretta. Questo può essere utile in diversi contesti, come ad esempio:

- Applicazioni che richiedono una grande quantità di risorse computazionali, come l'elaborazione di immagini o video ad alta definizione.
- Problemi che richiedono l'elaborazione di grandi quantità di dati, come la ricerca in un enorme database di informazioni.
- Problemi in cui la precisione non è essenziale e dove l'errore può essere tollerato, come nell'analisi statistica.

L'algoritmo di Las Vegas viene spesso utilizzato in combinazione con altri algoritmi per migliorare le prestazioni e ridurre i tempi di elaborazione; ad esempio, può essere utilizzato in combinazione con l'algoritmo di Monte Carlo per risolvere problemi di ricerca o di conteggio approssimato.

Un altro ambito di applicazione è inoltre quello crittografico, ad esempio, l'algoritmo RSA è un algoritmo di crittografia che si basa sulla difficoltà computazionale di fattorizzare grandi numeri primi. Poiché la fattorizzazione dei numeri primi richiede una grande quantità di risorse computazionali, l'algoritmo RSA può richiedere un tempo di elaborazione molto elevato. L'algoritmo di Las Vegas può essere utilizzato in questo contesto per accelerare l'elaborazione del sistema RSA.

In sintesi, l'algoritmo di Las Vegas è un algoritmo probabilistico che garantisce sempre una soluzione corretta, ma il tempo di esecuzione può variare notevolmente a causa della natura probabilistica dell'algoritmo, per questo viene utilizzato principalmente in situazioni in cui la precisione non è essenziale e dove il tempo di elaborazione può essere variabile, ma dove è comunque importante ottenere una soluzione corretta.

2. Quicksort

Quicksort è un algoritmo di ordinamento basato sulla tecnica "divide et impera". L'idea di base dell'algoritmo è di scegliere un elemento dell'array, chiamato pivot, e dividere l'array in due parti, in modo che tutti gli elementi nella prima parte siano minori o uguali al pivot e tutti gli elementi nella seconda parte siano maggiori del pivot. Quindi, l'algoritmo si ripete ricorsivamente su entrambe le parti dell'array, fino a quando l'array è completamente ordinato.

```
def quicksort(arr):
    if len(arr) <= 1:
        return arr
    else:
        pivot = arr[0]
        left = []
        right = []
        for i in range(1, len(arr)):
            if arr[i] < pivot:
                left.append(arr[i])
            else:
                right.append(arr[i])
        return quicksort(left) + [pivot] + quicksort(right)
```

In questa implementazione, la funzione "quicksort" prende in input un array e utilizza l'algoritmo di quicksort per ordinarlo. L'elemento di pivot viene scelto come il primo elemento dell'array. L'array viene quindi diviso in due sotto-array: uno con tutti gli elementi minori del pivot (left) e uno con tutti gli elementi maggiori o uguali al pivot (right). L'algoritmo si ripete ricorsivamente su entrambe le parti dell'array, fino a quando l'array è completamente ordinato.

L'algoritmo di Las Vegas può migliorare l'implementazione precedente dell'algoritmo di quicksort perché utilizza una scelta casuale dell'elemento di pivot. Nell'implementazione precedente, l'elemento di pivot viene scelto come il primo elemento dell'array, il che può portare a un tempo di esecuzione peggiore in alcuni casi particolari. Ad esempio, se l'array è già ordinato o quasi ordinato, l'elemento di pivot potrebbe essere il minimo o il massimo valore dell'array, il che porterebbe a una divisione sbilanciata dell'array e quindi a un tempo di esecuzione peggiore. Utilizzando l'algoritmo di Las Vegas, invece, l'elemento di pivot viene scelto casualmente, il che riduce la probabilità di scelte sbilanciate e può migliorare le prestazioni dell'algoritmo.

Ecco un esempio di implementazione dell'algoritmo di quicksort probabilistico:

```
import random

def quicksort_las_vegas(arr):
    if len(arr) <= 1:
        return arr
    else:
        pivot = random.choice(arr)
        left = [x for x in arr if x < pivot]
        middle = [x for x in arr if x == pivot]
        right = [x for x in arr if x > pivot]
        return quicksort_las_vegas(left) +
          middle +
          quicksort_las_vegas(right)
```

In questa implementazione, la funzione "quicksort_las_vegas" prende in input un array e utilizza l'algoritmo di quicksort probabilistico per ordinare l'array. L'elemento di pivot viene scelto casualmente utilizzando la funzione "random.choice", invece di essere scelto come il primo, l'ultimo o l'elemento mediano dell'array. Questo rende l'algoritmo di quicksort probabilistico.

Il resto della funzione "quicksort_las_vegas" è simile all'algoritmo di quicksort deterministico: l'array viene diviso in tre sotto-array (sinistra, centro e destra), ordinati separatamente e poi concatenati insieme per produrre l'array ordinato finale.

Questa implementazione dell'algoritmo di Las Vegas garantisce sempre una soluzione corretta, ma il tempo di esecuzione può variare notevolmente in base alla scelta casuale dell'elemento di pivot. Tuttavia, l'algoritmo di quicksort probabilistico è spesso più veloce dell'implementazione deterministica dell'algoritmo di quicksort.

Dal punto di vista della complessità computazionale, l'algoritmo di quicksort con l'algoritmo di Las Vegas ha la stessa complessità dell'algoritmo di quicksort deterministico. La complessità computazionale dell'algoritmo di quicksort, in media, è $O(n \log n)$, dove n è la dimensione dell'array da ordinare.

Tuttavia, l'implementazione dell'algoritmo di Las Vegas può ridurre la probabilità di casi sfavorevoli che possono aumentare il tempo di esecuzione dell'algoritmo di quicksort deterministico. Ad esempio, se l'elemento di pivot viene scelto casualmente, la probabilità che l'array venga diviso in modo sbilanciato (con una delle due parti molto più grande dell'altra) è molto bassa. Ciò può ridurre il tempo di esecuzione dell'algoritmo in alcuni casi particolari.

In generale, l'algoritmo di Las Vegas può fornire prestazioni migliori in casi particolari, ma non è garantito che fornisca sempre prestazioni migliori rispetto all'algoritmo di quicksort deterministico.

La scelta dell'algoritmo da utilizzare dipende quindi dalle esigenze specifiche dell'applicazione e dalla natura dei dati da ordinare.

3. Selezione del mediano

Ecco un esempio di implementazione dell'algoritmo di selezione del mediano senza l'utilizzo dell'algoritmo di Las Vegas in Python:

```
def select_median(arr):
    if len(arr) == 1:
        return arr[0]
    else:
        pivot = arr[0]
        left = [x for x in arr if x < pivot]
        right = [x for x in arr if x > pivot]
        k = len(left)
        if len(left) == 0:
            return pivot
        elif k == len(arr):
            return arr[k-1]
        else:
            return select_median(left) if k <= len(arr)/2 else
select_median(right)
```

In questa implementazione, la funzione "select_median" prende in input un array e utilizza l'algoritmo di selezione del mediano per trovare il mediano dell'array. L'elemento di pivot viene scelto come il primo elemento dell'array.

La funzione "select_median" si ripete ricorsivamente su uno dei due sotto-array, in base alla posizione del mediano rispetto all'elemento di pivot. Tuttavia, questa implementazione potrebbe avere prestazioni peggiori in alcuni casi particolari, ad esempio quando l'array è già ordinato o quasi ordinato. In tali casi, l'elemento di pivot potrebbe essere il minimo o il massimo valore dell'array, il che porterebbe a una divisione sbilanciata dell'array e quindi a un tempo di esecuzione peggiore.

L'utilizzo dell'algoritmo di Las Vegas nella scelta casuale dell'elemento di pivot riduce la probabilità di scelte sbilanciate e può migliorare le prestazioni dell'algoritmo in questi casi particolari.

Ecco un esempio di implementazione dell'algoritmo di selezione del mediano con l'algoritmo di Las Vegas in Python:

```
import random

def select_median(arr):
    if len(arr) == 1:
        return arr[0]
    else:
        pivot = random.choice(arr)
```

```
        left = [x for x in arr if x < pivot]
        right = [x for x in arr if x > pivot]
        k = len(left)
        if len(left) == 0:
            return pivot
        elif k == len(arr):
            return arr[k-1]
        else:
            return   select_median(left)   if   k   <=   len(arr)/2   else
select_median(right)
```

In questa implementazione, la funzione "select_median" prende in input un array e utilizza l'algoritmo di selezione del mediano con l'algoritmo di Las Vegas per trovare il mediano dell'array. L'elemento di pivot viene scelto casualmente utilizzando la funzione "random.choice", invece di essere scelto come il primo, l'ultimo o l'elemento mediano dell'array.

La funzione "select_median" si ripete ricorsivamente su uno dei due sotto-array, in base alla posizione del mediano rispetto all'elemento di pivot. In questo modo, l'algoritmo cerca di eliminare metà degli elementi dell'array ad ogni iterazione, fino a quando viene trovato il mediano.

L'implementazione dell'algoritmo di Las Vegas nella scelta casuale dell'elemento di pivot riduce la probabilità di scelte sbilanciate e può migliorare le prestazioni dell'algoritmo in casi particolari. Tuttavia, la complessità computazionale dell'algoritmo di selezione del mediano con l'algoritmo di Las Vegas non è garantita essere migliore dell'implementazione deterministica dell'algoritmo.

Ecco un esempio di implementazione dell'algoritmo di ricerca di un elemento in un albero binario di ricerca bilanciato senza l'utilizzo dell'algoritmo di Las Vegas in Python:

```
class TreeNode:
    def __init__(self, val):
        self.val = val
        self.left = None
        self.right = None

def search_bst(root, val):
    if not root:
        return None
    if root.val == val:
        return root
    elif root.val > val:
        return search_bst(root.left, val)
    else:
        return search_bst(root.right, val)
```

In questa implementazione, la funzione "search_bst" prende in input un nodo radice "root" e un valore "val" da cercare nell'albero binario di ricerca. L'algoritmo esplora l'albero partendo dalla radice e seguendo il percorso corretto a seconda del valore dell'elemento. L'algoritmo termina quando viene trovato l'elemento desiderato o quando si raggiunge una foglia dell'albero senza trovare l'elemento.

Ecco un esempio di implementazione dell'algoritmo di ricerca di un elemento in un albero binario di ricerca bilanciato con l'utilizzo dell'algoritmo di Las Vegas in Python:

```
import random

class TreeNode:
    def __init__(self, val):
        self.val = val
        self.left = None
        self.right = None

def search_bst(root, val):
    if not root:
        return None
    if root.val == val:
        return root
    elif root.val > val:
        if random.random() < 0.5:
            return search_bst(root.left, val)
        else:
            return search_bst(root.right, val)
    else:
        if random.random() < 0.5:
            return search_bst(root.right, val)
        else:
            return search_bst(root.left, val)
```

In questa implementazione, la funzione "search_bst" prende in input un nodo radice "root" e un valore "val" da cercare nell'albero binario di ricerca. L'algoritmo utilizza la funzione "random.random" per scegliere casualmente se cercare l'elemento a sinistra o a destra in base al valore della probabilità di 0.5. Questa scelta casuale riduce la probabilità che l'algoritmo debba visitare molti nodi dell'albero per trovare l'elemento desiderato.

L'algoritmo di ricerca di un elemento in un albero binario di ricerca bilanciato con l'utilizzo dell'algoritmo di Las Vegas garantisce sempre la correttezza della soluzione restituita.

4. Conclusione

In conclusione, gli algoritmi probabilistici come Monte Carlo e Las Vegas sono utilizzati in numerose applicazioni, dall'analisi di dati alla criptografia, e possono offrire prestazioni migliori rispetto agli algoritmi deterministici in alcuni casi particolari. Tuttavia, come abbiamo visto, gli algoritmi probabilistici non garantiscono sempre di trovare la soluzione corretta e possono richiedere più tempo di esecuzione rispetto agli algoritmi deterministici in alcuni casi.

Il confronto tra Monte Carlo e Las Vegas dipende dalle specifiche applicazioni e dalle esigenze dell'utente. In generale, Monte Carlo è utilizzato per la risoluzione di problemi numerici, come il calcolo di integrali e la simulazione di fenomeni fisici, mentre Las Vegas è utilizzato per la risoluzione di problemi di ricerca, come la ricerca di un elemento in una struttura dati.

Inoltre, gli algoritmi di Las Vegas possono essere più efficienti degli algoritmi Monte Carlo in quanto cercano di limitare il numero di iterazioni necessarie per trovare la soluzione corretta. Tuttavia, gli algoritmi di Las Vegas richiedono spesso una scelta casuale durante l'esecuzione, che può portare a risultati diversi ogni volta che l'algoritmo viene eseguito.

Più in generale, la scelta tra l'utilizzo di algoritmi probabilistici e algoritmi deterministici dipende dalle esigenze dell'applicazione e dalle specifiche del problema da risolvere.

Esistono alcune best practice che possono essere utilizzate per determinare se un algoritmo deterministico o probabilistico è più adatto per una determinata applicazione o problema.

- gli algoritmi deterministici sono preferiti quando la soluzione esatta del problema può essere ottenuta con certezza e senza ambiguità. Gli algoritmi deterministici sono spesso utilizzati per risolvere problemi di ottimizzazione, ricerca e ordinamento.
- gli algoritmi probabilistici sono preferiti quando la soluzione esatta del problema è difficile o impossibile da ottenere con certezza, ma una soluzione approssimata può essere sufficiente per le esigenze dell'applicazione. Gli algoritmi probabilistici sono spesso utilizzati per la simulazione di fenomeni fisici, la ricerca di informazioni su Internet e la criptografia.

In generale, quando si sceglie tra algoritmi deterministici e algoritmi probabilistici, è importante considerare le esigenze dell'applicazione, il tempo di esecuzione richiesto e la precisione della soluzione richiesta. In alcuni casi, può essere opportuno utilizzare una combinazione di algoritmi deterministici e probabilistici per ottenere la soluzione migliore possibile.

Bibliografia

- Alan Bertossi, Alberto Motresor: Algoritmi e strutture di dati, Città Studi Edizioni, terza edizione;
- C. Demetrescu, I. Finocchi, G. F. Italiano: Algoritmi e strutture dati, McGraw-Hill, seconda edizione;
- Crescenzi, Gambosi, Grossi: Strutture di Dati e Algoritmi, Pearson/Addison-Wesley;
- Sedgewick: Algoritmi in C, Pearson, 2015;
- Cormen Leiserson Rivest Stein-Introduzione Agli Algoritmi E Strutture Dati-Prima Edizione.