Luiss
Libera Università Internazionale
degli Studi Sociali Guido Carli

Corso di preparazione per la selezione territoriale delle Olimpiadi di Informatica

Euristiche & Stringhe

Blerina Sinaimeri & Nicola Prezza

13 maggio 2021

LUISS

E finalmente… il contest!

15 Maggio

- Dovrete accedere alla piattaforma utilizzando la password generata al momento della registrazione (la stessa password che usate per Mattermost)
- Se qualcuno ha cambiato la password in Mattermost, contattate un tutor
- Il concorso inizia a 15 Maggio 00:00 e dura 24 ore
- C'è una finestra di 5 ore dal primo accesso alla piattaforma
- Tra pochi giorni pubblicheremo il link alla piattaforma del concorso e ulteriori istruzioni nella sezione degli annunci su Mattermost

Certificati di frequenza

- I certificati di frequenza del corso sarà all'indirizzo email indicato nel momento della registrazione al corso.

Programma di oggi

1. Soluzione di alcuni esercizi assegnati la scorsa settimana
2. Euristiche
3. Un esempio usando il problema del cammino minimo
   - Un problema dalle OIS (2020): Multi-Layer Dictionary (dictionary)
4. Strutture dati per stringhe e sequenze
   - std::string
   - Liste concatenate
   - Tries
5. Un problema di competitive programming: join strings
6. Ulteriori problemi su stringhe

Esercizi dalla scorsa settimana

- Il tesoro del pirata Barbablú (Selezioni territoriali 2012)
  https://training.olinfo.it/ - /task/barbablu/statement
- Alberi (OII 2003)
  https://training.olinfo.it/#/task/alberi/statement
- Flood forecasting (OIS 2020)
  https://training.olinfo.it/#/task/ois_rainstorm/statement
- Paper (ABC 2017 – Algorithm Bergamo Contest)
  https://training.olinfo.it/#/task/abc_paper/statement
- Islands (IOI 2008)
  https://training.olinfo.it/#/task/islands/statement

Euristica

Eurisiko= io trovo

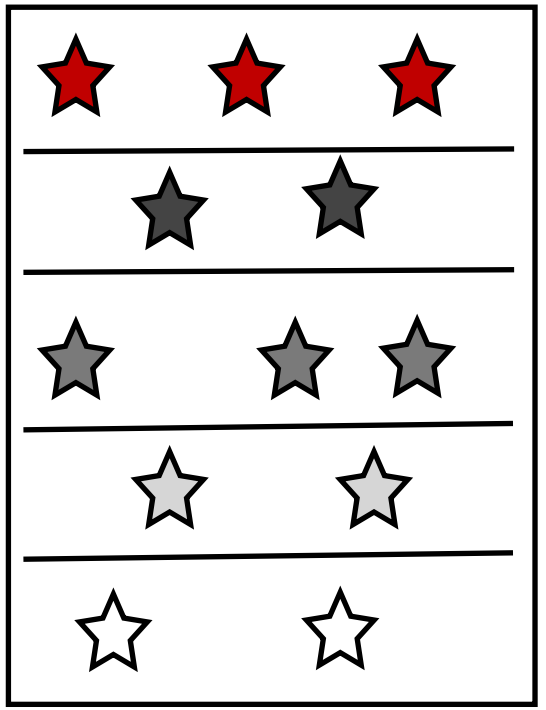
- E una parola di derivazione greca
- ispirata al celebre aneddoto su Archimede e la corona d'oro

Euristica

- Un’euristica è un insieme di considerazioni di buon senso!
- Un’euristica non garantisce la soluzione!
- Allora a cosa serve?
- "costa” molto meno di un algoritmo corretto:
  - Tempo
  - Spazio
- fornisce “spesso” una soluzione “vicina” a quella corretta:
  - Questo si può misurare ad esempio tramite una metrica, come “distanza soddisfacente tra soluzioni”

- Imaginiamo le soluzioni raggrupate secondo una misura di qualità.

Soluzioni raggrupate in ordine crescent della qualita



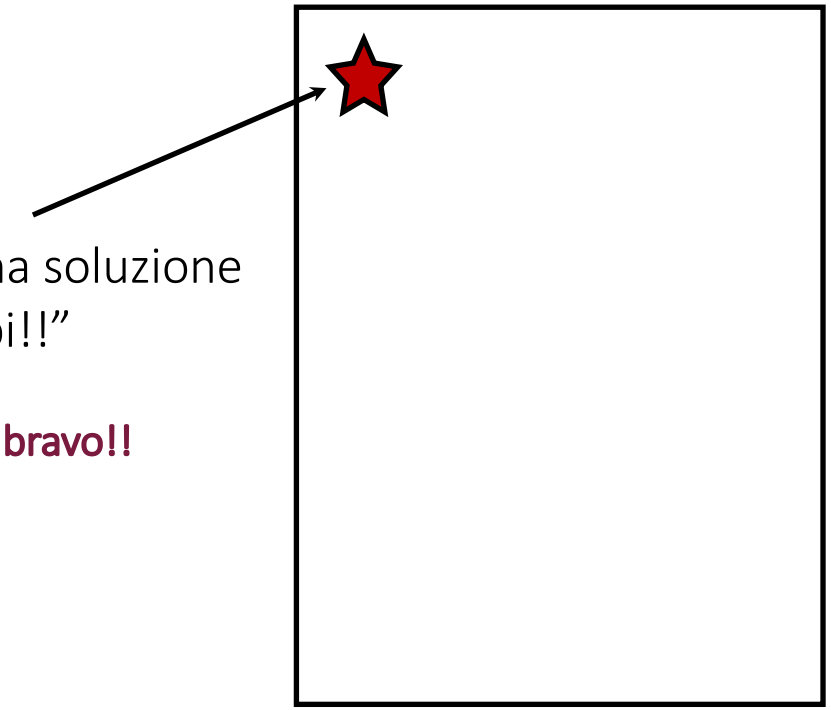
Soluzioni costo ottimale

- Imaginiamo le soluzioni raggrupate secondo una misura di qualità.

Un algoritmo esatto garantisce sempre che la soluzione prodotta è ottima!!

"Ho trovato una soluzione ottimale! Yuppi!!"

In effetti bravo!!

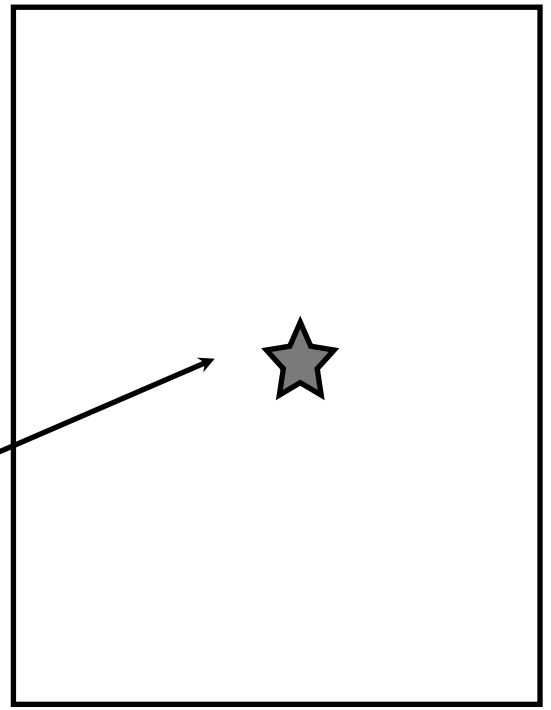


Soluzioni costo ottimale

- Imaginiamo le soluzioni raggrupate secondo una misura di qualità.

Un euristica produce solo una soluzione e non da nessuna informazione sul quanto sia lontana dalla soluzione ottima!

"Ho trovato una soluzione che costa 100! Yuppi!!"

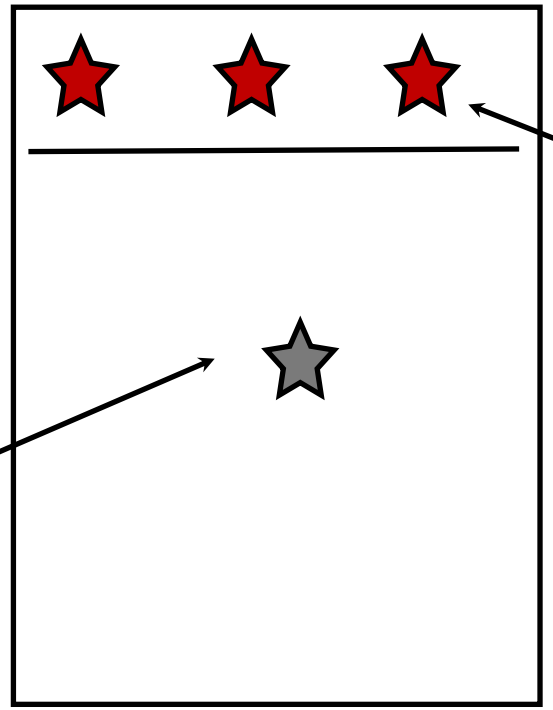


Soluzioni costo ottimale

- Imaginiamo le soluzioni raggrupate secondo una misura di qualità.

Un euristica produce solo una soluzione e non da nessuna informazione sul quanto sia lontana dalla soluzione ottima!

"Ho trovato una soluzione che costa 100! Yuppi!!"

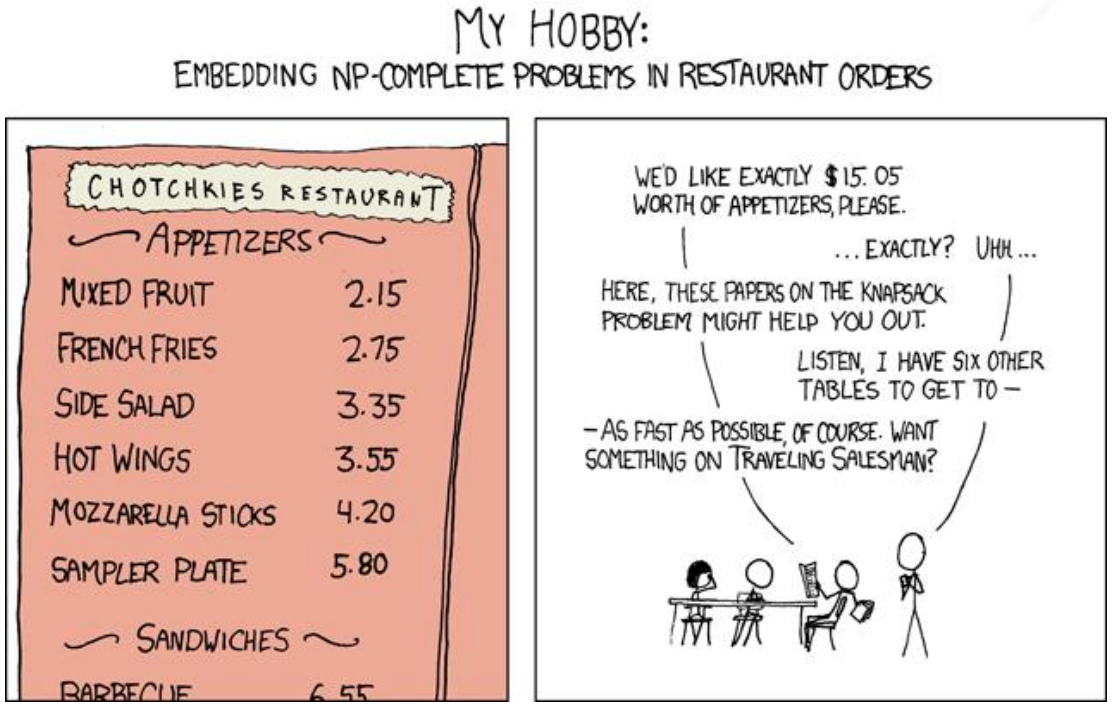


Soluzioni costo ottimale

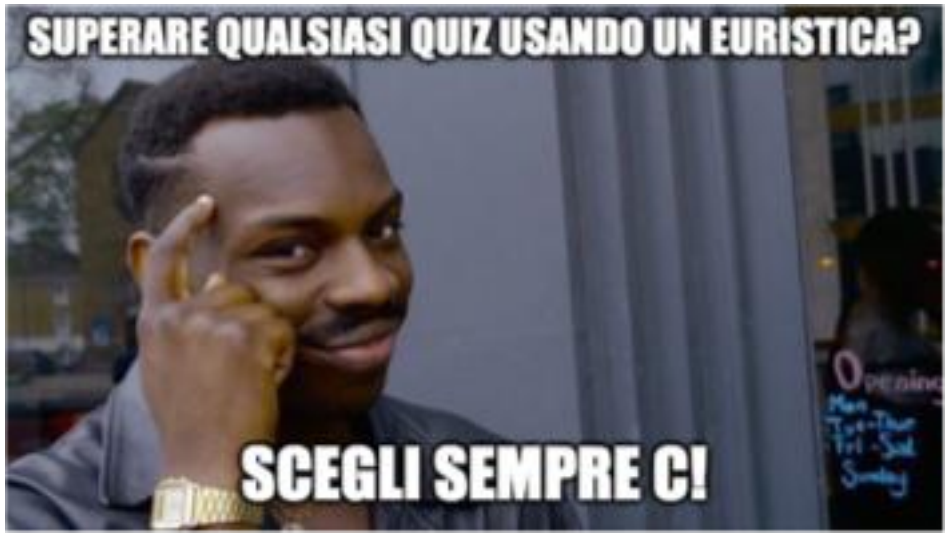
Se la soluzione ottimale costa 95 allora si, è una buona soluzione. Ma se la soluzione ottimale costa 2 allora è una pessima soluzione!

Applicazioni

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9



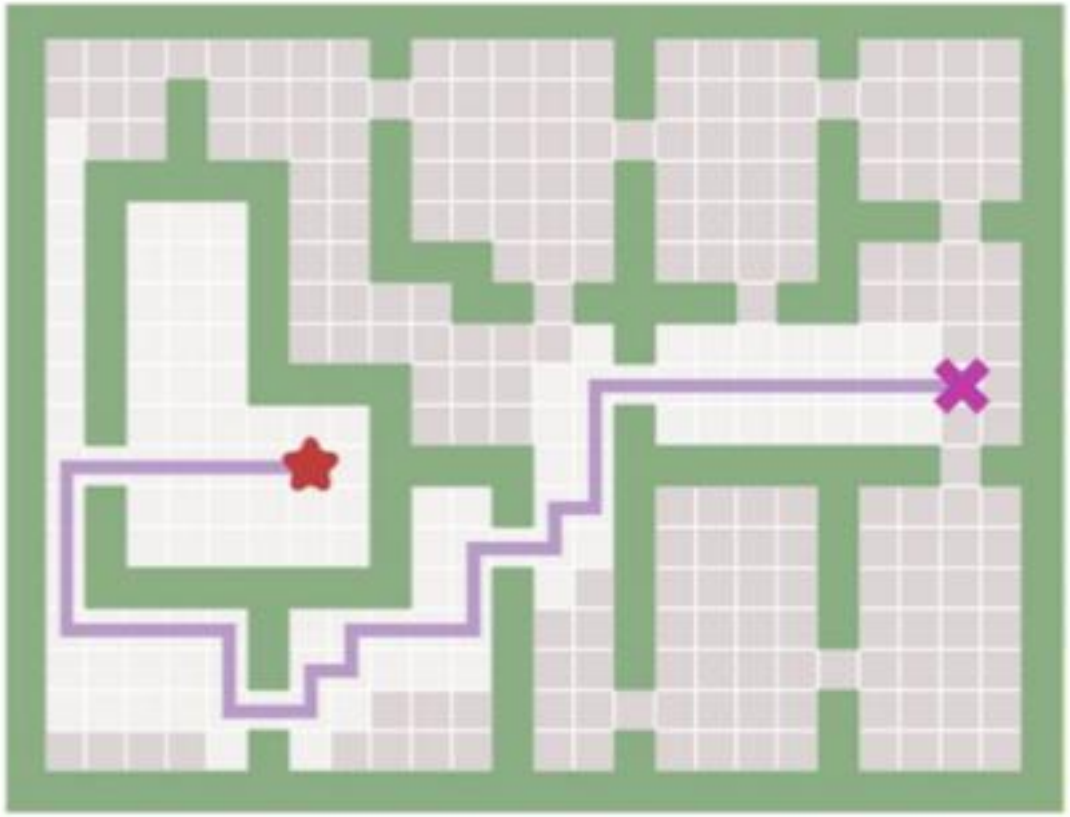
Altre applicazioni ;)



Il problema del cammino minimo

- Avete visto la scorsa volta il problema del cammino minimo.

Problema: Trovare il cammino minimo da ★ a ✕

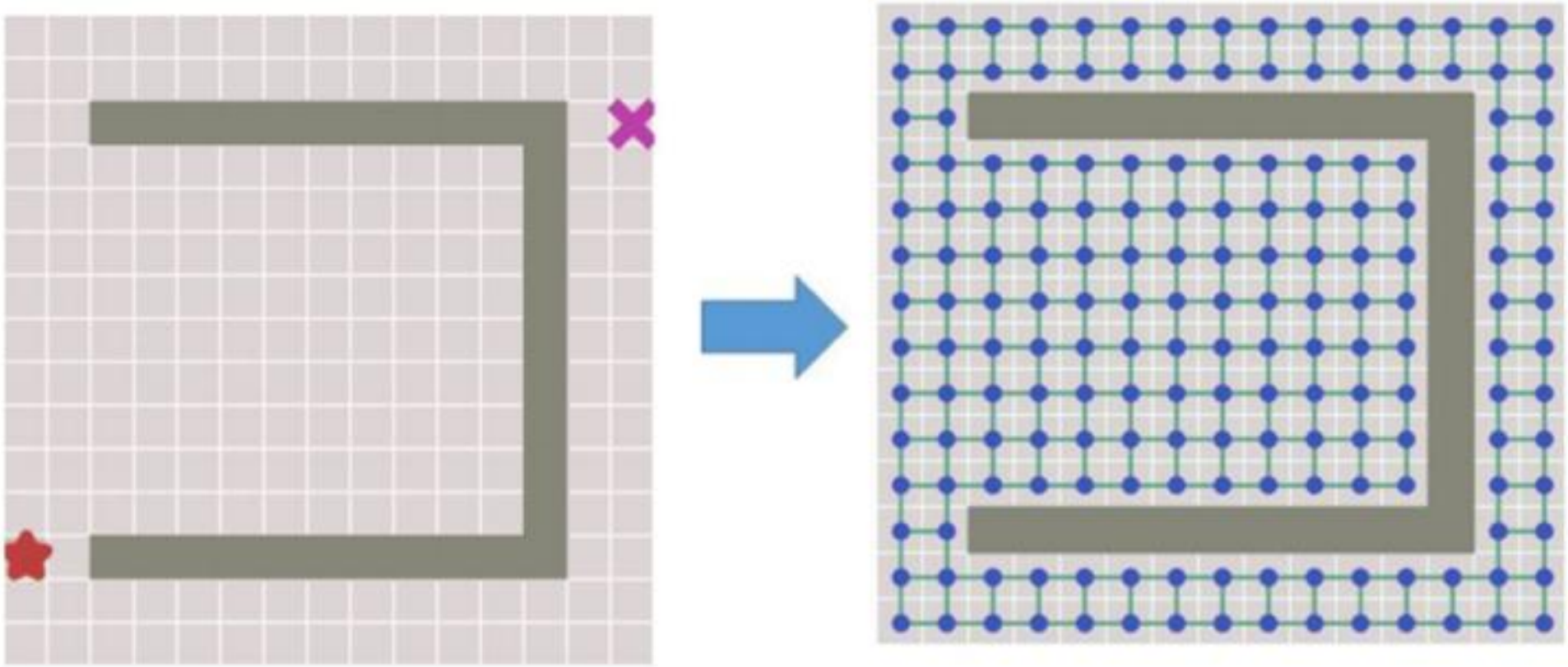


Come rappresentare un labirinto come un grafo?

- Ogni cella può essere rappresentata con un nodo

- I muri non sono connessi a nulla

Un esempio di labirinto



Userò in seguito delle belle animazioni che illustranno le BFS/DFS

https://cs.stanford.edu/people/abisee/tutorial/

https://cs.stanford.edu/people/abisee/tutorial/bfs.html

https://cs.stanford.edu/people/abisee/tutorial/dfs.html

https://cs.stanford.edu/people/abisee/tutorial/bfsdfs.html

Potete aggiungere muri e spostare i punti di partenza e arrivo.

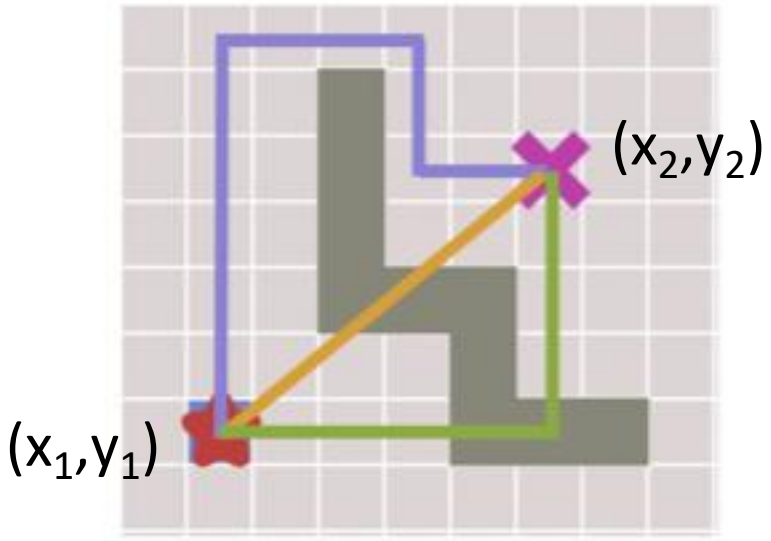
Problema:

- Come velocizzare la BFS per trovare prima il cammino minimo tra minimo da ★ a ✕ ?

Problema:

- Come velocizzare la BFS per trovare prima il cammino minimo tra minimo da ★ a ?

- IDEA Greedy: Invece di considerare tutti i vertici adjacenti ad un vertice considero SOLO quello che mi avvicina al obiettivo.
  - Come definire « vicino » ? Ricordiamo che un euristica deve essere facile da calcolare, siccome siamo su un piano possiamo avere coordinate e usare la distanza Euclidea nel piano.

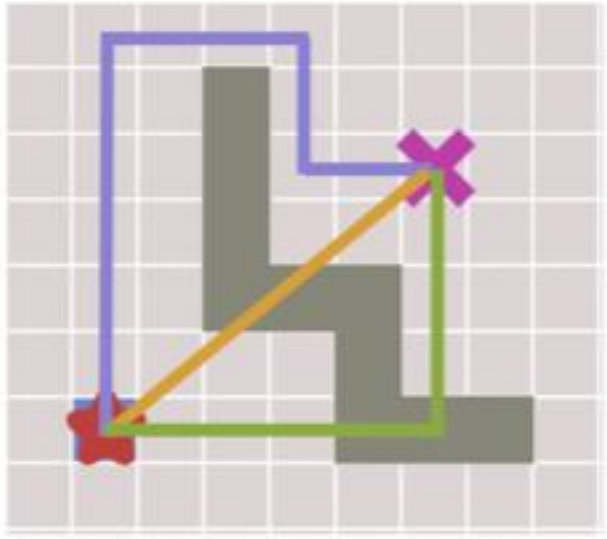


$$d = \sqrt[2]{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

Problema:

- Come velocizzare la BFS per trovare prima il cammino minimo tra minimo da ★ a ⚔ ?

- IDEA Greedy: Invece di considerare tutti i vertici adjacenti ad un vertice considero SOLO quello che mi avvicina al obiettivo.
  - Come definire « vicino » ? Ricordiamo che un euristica deve essere facile da calcolare, siccome siamo su un piano possiamo avere coordinate e usare la distanza Euclidea nel piano.

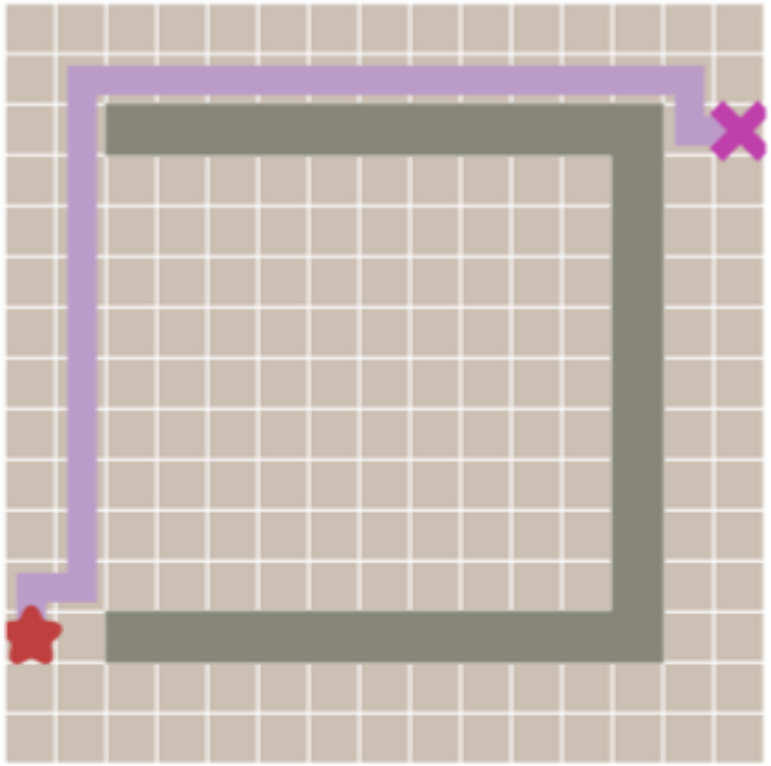


- Heuristica: GREEDY BEST FIRST

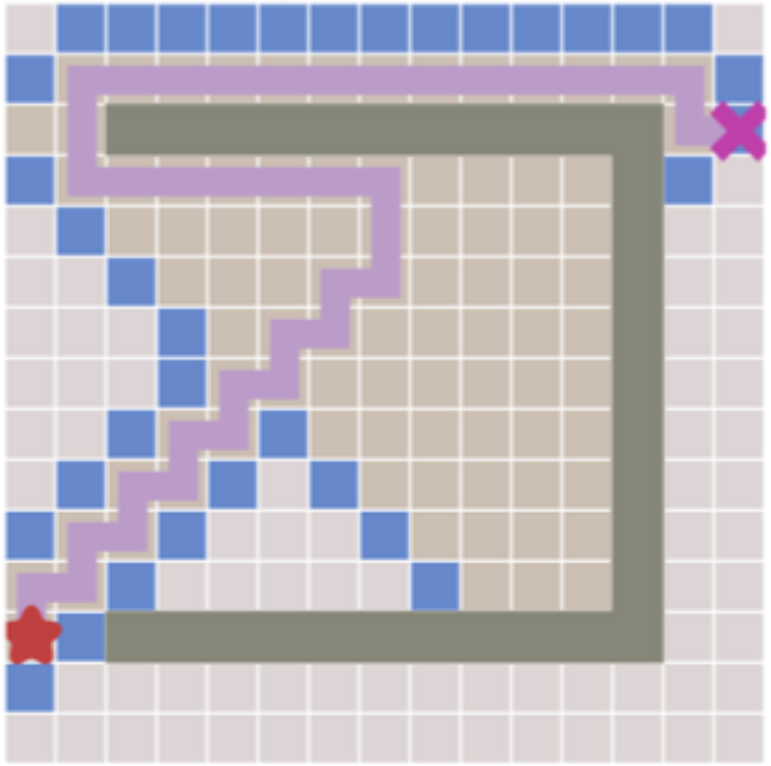
https://cs.stanford.edu/people/abisee/tutorial/greedy.html

Greedy Best First Search è certamente più veloce della BFS ma trova sempre il cammino minimo?

Breadth First Search

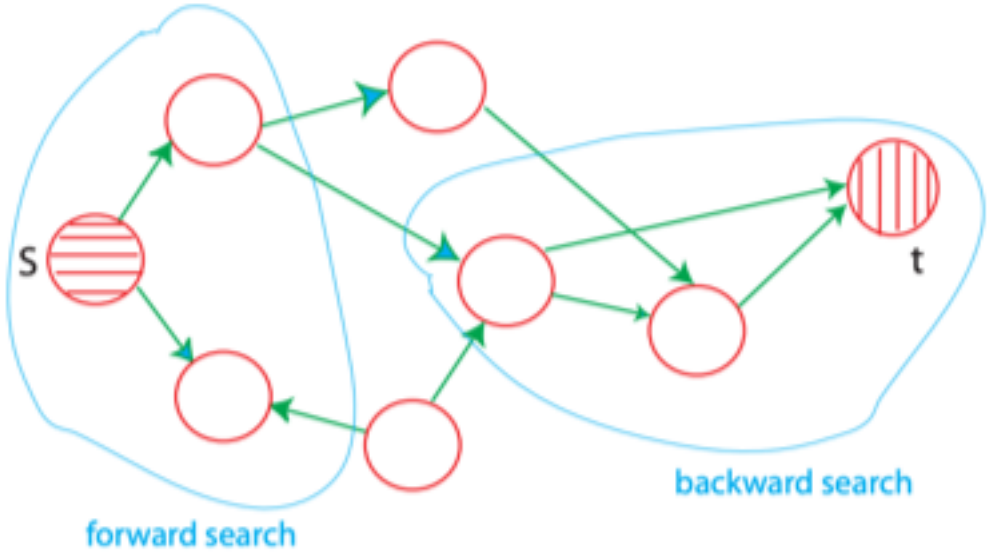


Greedy Best-First Search

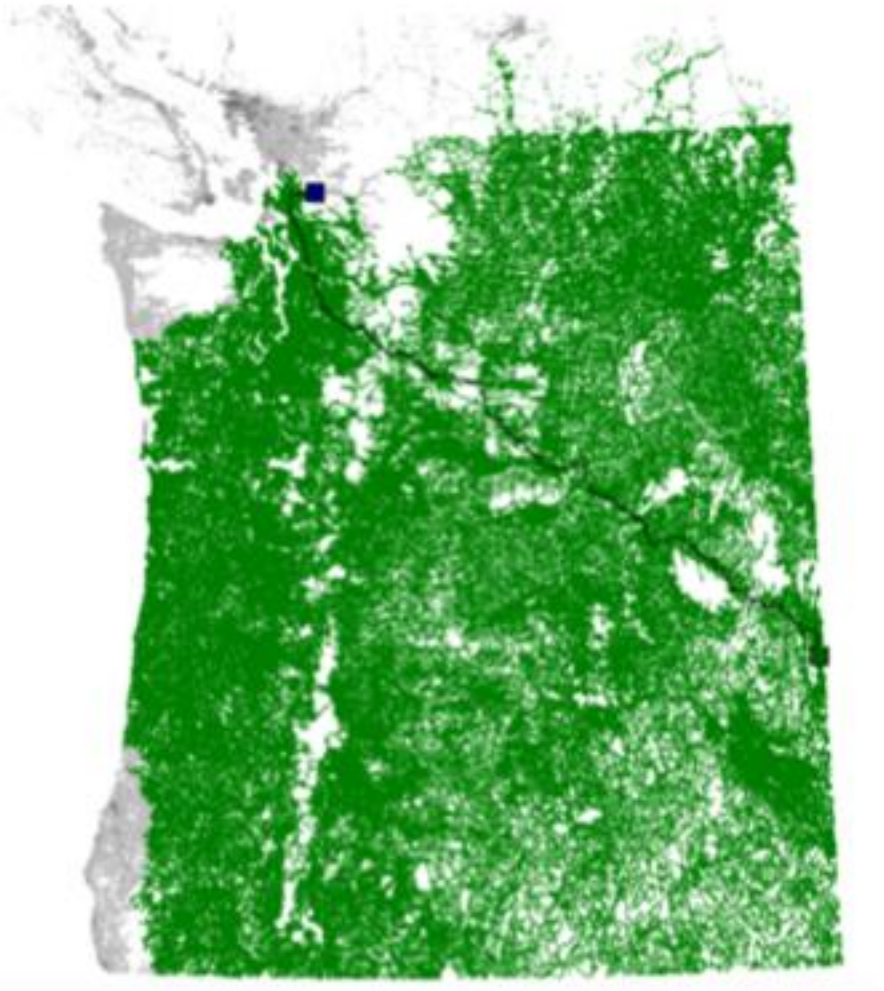


Search bidirezionale

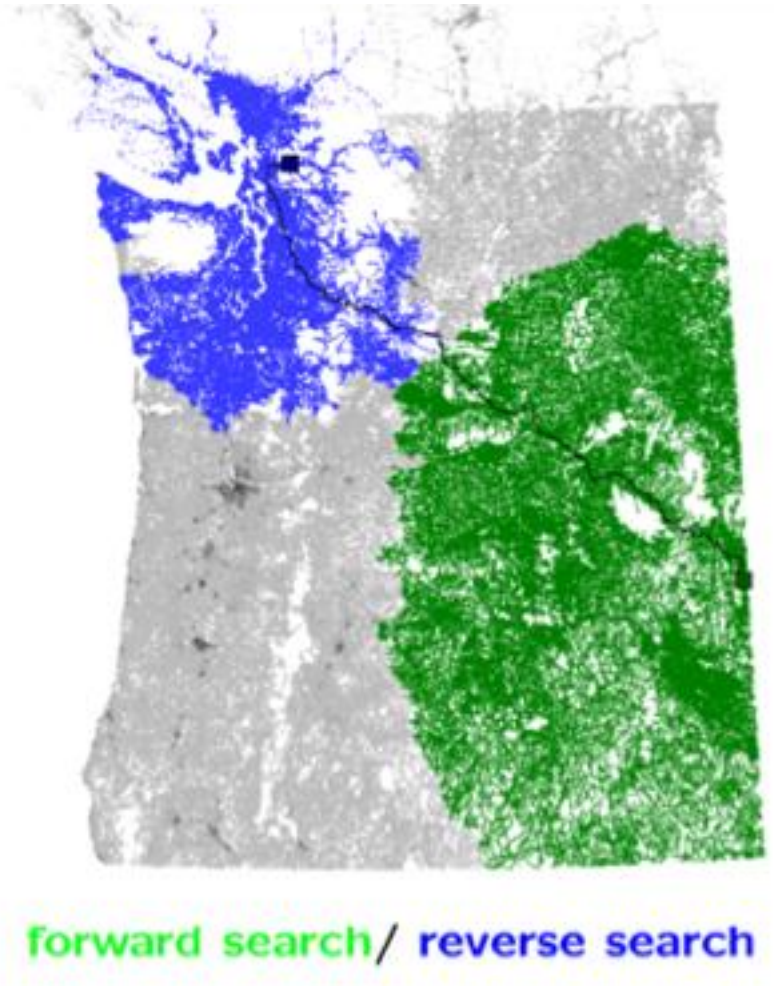
Search bi-direzionale



Algoritmo di Dijkstra



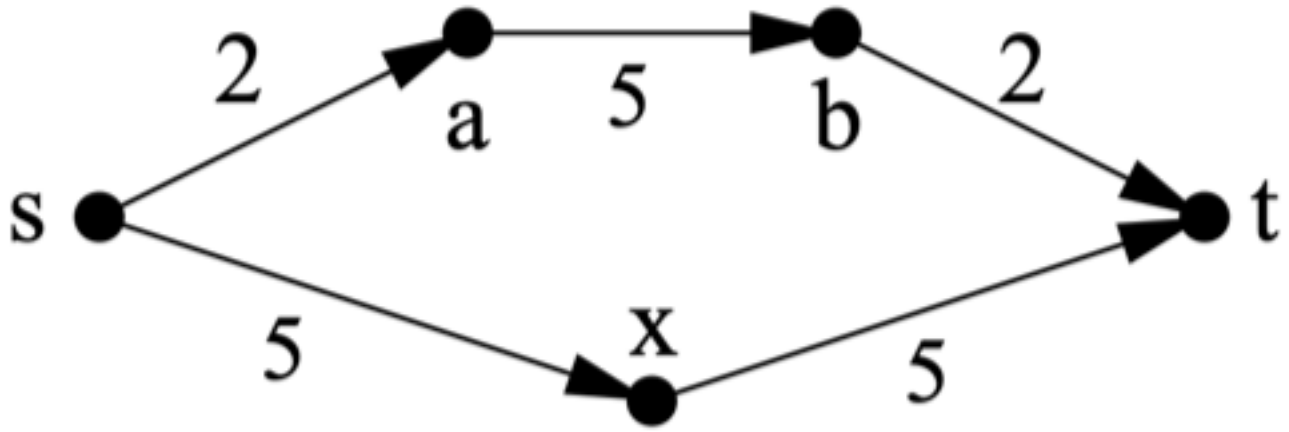
Algoritmo di Dijkstra bidirezionale



Dijkstra vs Bidirectional Dijkstra

https://www.youtube.com/watch?v=8Jjdp6f7oaE

L’algoritmo non è cosi semplice



Esercizi olimpiadi

Multi-Layer Dictionary (dictionary):
https://training.olinfo.it/#/task/ois_dictionary/statement

Multi-Layer Dictionary (`dictionary`)

When William encounters a new English word, he usually looks for its definition in the dictionary. For example, a definition for the word "unencumbered" is as follows:

unencumbered = not having any burden

Sometimes this process creates a recursion, because William might then need to look up the definition for "burden", which again might be using new words that William doesn't know yet.

To avoid encountering this issue, you should know that every dictionary has a *defining vocabulary*: a fixed list of words that was used to write the dictionary's definitions. For example, the *Longman Dictionary of Contemporary English* uses a defining vocabulary of approximately 2000 words, while the *Oxford Advanced Learner's Dictionary* uses one of approximately 3000 words. If you learn all of those beforehand then you can safely read the dictionary without encountering unexpected words.

Learning a couple thousand words just to be able to use a dictionary is too much to ask of William: being lazy, he always looks for an easier way. In fact, he found a (freely available online) dictionary called **Learn These Words First**[1]. This dictionary is special because it's *layered*:

- There is a first layer that contains just 61 words (called *primitive words*) which are mostly universal concepts, explained through pictures.
- Each successive layer builds upon the previous layers by defining some new words using *only the words that are already defined in the previous layers.*

This means that by reading the dictionary in the correct order (level by level) you will never encounter unexpected words, and instead of thousands of primitive words you only need to learn 61 of them!

The discovery of this dictionary led William to wonder about a challenging new problem: given a dictionary, find the smallest possible set of primitive words that you need to know in order to be able to recursively define every other word of the dictionary (either by using the initial words or by using words derived from other definitions).

☞ Among the attachments of this task you may find a template file `dictionary.*` with a sample incomplete implementation.

Input

The first line contains the integer N, the *total number of distinct words* used in the dictionary (including the words being defined). The second line contains the integer D, the number of definitions in the dictionary. Each of the next D lines contains one definition of a word: each definition starts with the word being defined, then an integer K, and then K words that form its definition.

Output

You need to write a line with an integer W, as small as you can, followed by W lines with words able to define all other words (in any order).

Constraints

- $1 \leq D \leq N \leq 1000$.
- Each word uses between 1 and 20 lowercase alphabetical letters.
- Some words in definitions may not have a definition themselves: you need to learn them first!

Scoring

This is a **partial score** task. Your program will be tested against several testcases and it will receive a score based on whether your proposed solution is valid and minimal. There will be 10 testcases (divided in 10 different subtasks, so the score of each testcase is independent from the others, and the max score for each testcase will be taken over all submissions). The first subtask contains the sample testcases, so in total there are 11 subtasks. For each testcase, the score will be:

- 0 if the proposed solution is not valid (i.e. if the output is malformed or if the selected words are not enough to recursively define all of the dictionary's words).
- $\min\left\{1, 0.2 + 0.8\left(\frac{N-W}{N-W_{ref}}\right)^3\right\}$ where W_{ref} is a reference result for that input.

Examples

input	output
7 3 cat 3 not a dog dog 3 a domesticated mammal human 2 a mammal	4 not a domesticated mammal
3 2 easy 2 not difficult difficult 2 not easy	2 not easy
4 4 a 1 b b 1 c c 1 d d 1 a	1 c

Explanation

In the **first sample case** by learning those 4 words we can define "human" and "dog". After that, since we know know the word "dog", we can define "cat" as well (without increasing the set of initial words).

In the **second sample case** there are two possible solutions: "not" + "easy", or "not" + "difficult".

In the **third sample case** knowing *any* word is enough.

Alcuni esercizi OII che richiedono tecniche viste oggi:

- Full-Body Workout (workout): https://training.olinfo.it/ - /task/ois_workout/statement
- Master Chef (kitchen): https://training.olinfo.it/ - /task/ois_kitchen/statement


OH UNA STELLA CADENTE,
FAMMI ESPRIMERE UN DESIDERIO
VORREI VINCERE LE
OLIMPIADI SENZA ESERCITARMI

Stringhe

Nicola Prezza

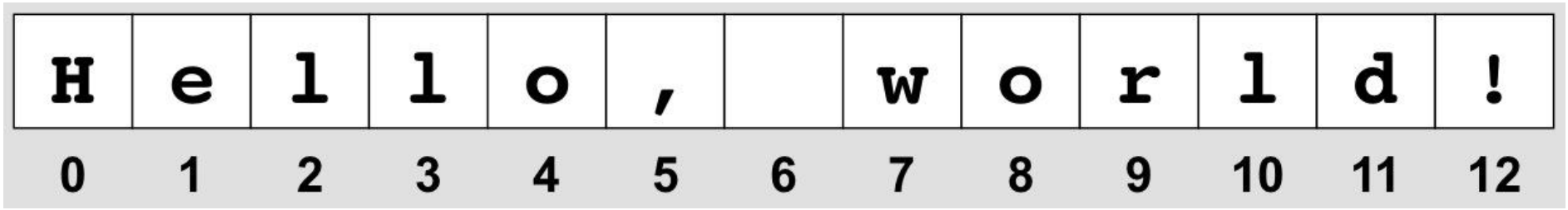
Programma di oggi

1. Strutture dati per stringhe e sequenze
   - std::string
   - Liste concatenate
   - Tries
2. Un problema di competitive programming: join strings
3. Ulteriori problemi su stringhe

Stringhe

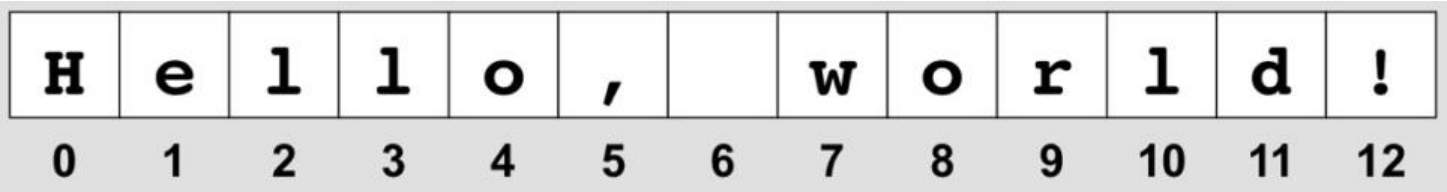
Stringhe

Una stringa è una sequenza di caratteri. Ogni carattere è associato ad una posizione (contiamo da 0)



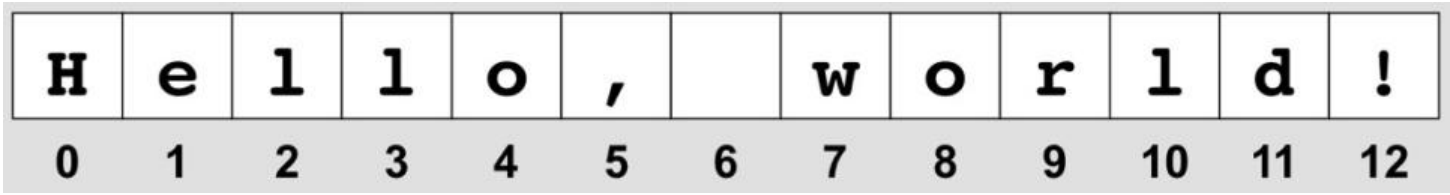
Per affrontare un problema su stringhe, è fondamentale saper usare la **struttura dati** giusta!

Stringhe



Una struttura dati è il modo di organizzare i dati (in questo caso, i caratteri della stringa) in memoria

Stringhe

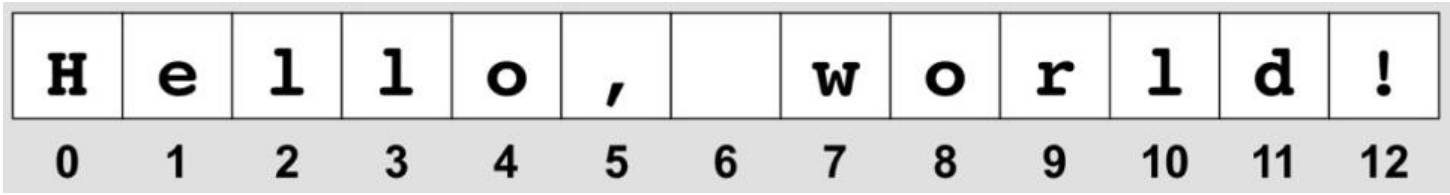


Una struttura dati è il modo di organizzare i dati (in questo caso, i caratteri della stringa) in memoria

Come scegliere la struttura?

1. identificate che operazioni andrete ad eseguire sulla stringa:
   - **accesso** random a caratteri / estrazione della stringa intera
   - **concatenazione** di stringhe
   - **edits** (cancellazioni, inserimenti)
   - **sostituzioni** di caratteri

Stringhe



Una struttura dati è il modo di organizzare i dati (in questo caso, i caratteri della stringa) in memoria

Come scegliere la struttura?

1. identificate che operazioni andrete ad eseguire sulla stringa:
   - **accesso** random a caratteri / estrazione della stringa intera
   - **concatenazione** di stringhe
   - **edits** (cancellazioni, inserimenti)
   - **sostituzioni** di caratteri

2. Scegliere la struttura che implementa queste operazioni il più velocemente possibile

Operazioni su stringhe

s = «hello»
t = « world!»

- **accesso**: s[4] (restituisce ‘o’)
- **estrai(s)** (restituisce «hello»)
- **concatenazione**: s + t (restituisce «hello world!»)
- **Inserimenti**: s.insert(2,’e’) (modifica s in «heello»)
- **cancellazione**: s.erase(3) (modifica s in «helo»)
- **sostituzioni**: s[1] = ‘a’ (modifica s in «hallo»)

std::string

In C++, std::string è un array di caratteri. Supporta le seguenti operazioni molto velocemente:

- accesso ad un carattere: s[i]
- aggiunta di un carattere in fondo: s.push_back(c)
- sostituzione di un carattere: s[i] = ‘a’
- estrazione della stringa intera

```
1  // string::operator[]
2  #include <iostream>
3  #include <string>
4
5  int main ()
6  {
7    std::string str ("Test string");
8    for (int i=0; i<str.length(); ++i)
9    {
10     std::cout << str[i];
11   }
12   return 0;
13 }
```

std::string

In C++, std::string è un array di caratteri. Supporta le seguenti operazioni molto velocemente (ottimale):

- accesso ad un carattere: s[i] O(1)
- aggiunta di un carattere in fondo: s.push_back(c) O(1)
- sostituzione di un carattere: s[i] = ‘a’ O(1)
- estrazione della stringa intera O(n)

Dove n è la lunghezza della stringa

```
1  // string::operator[]
2  #include <iostream>
3  #include <string>
4
5  int main ()
6  {
7    std::string str ("Test string");
8    for (int i=0; i<str.length(); ++i)
9    {
10     std::cout << str[i];
11   }
12   return 0;
13 }
```

Attenzione: le seguenti operazioni sono supportate, ma sono **lente**!

- std::string::insert
- std::string::erase
- concatenazione di stringe: string x = z + w

Complessità: O(n). Perchè?

Attenzione: le seguenti operazioni sono supportate, ma sono **lente**!

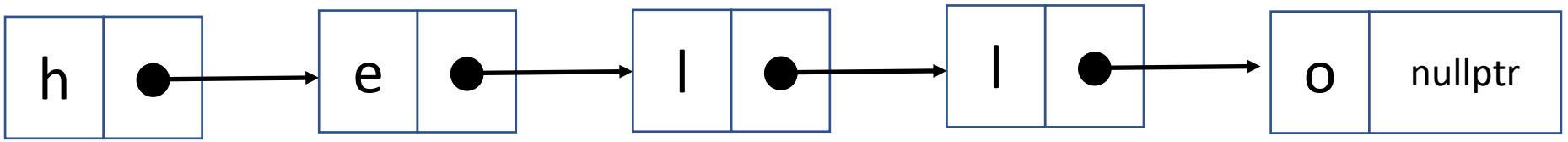
- std::string::insert
- std::string::erase
- concatenazione di stringe: string x = z + w

Complessità: O(n). Perchè? le stringhe sono implementate come array di char, quindi queste operazioni richiedono di «spostare» intere porzioni di questi array.

linked lists

liste concatenate

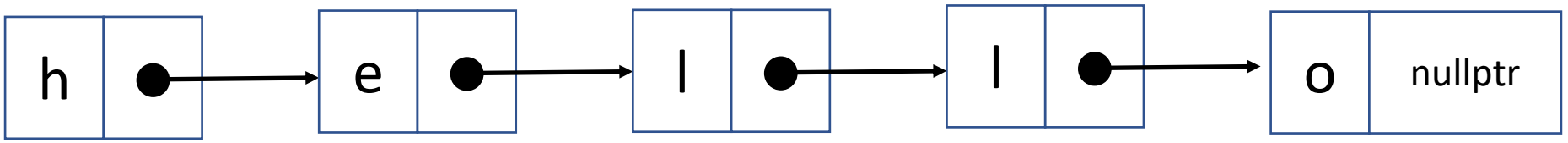
Una lista è una «catena» di celle, ognuna contenente un carattere e un puntatore alla prossima cella



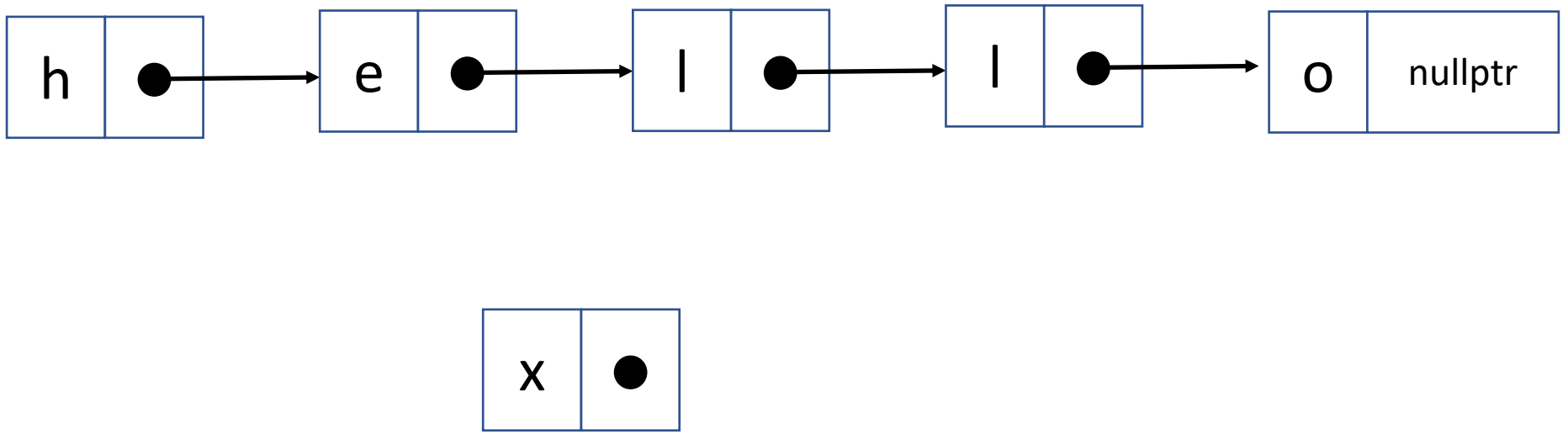
Questa lista codifica la stringa «hello»

L'ultima cella non punta a niente (in C++: nullptr)

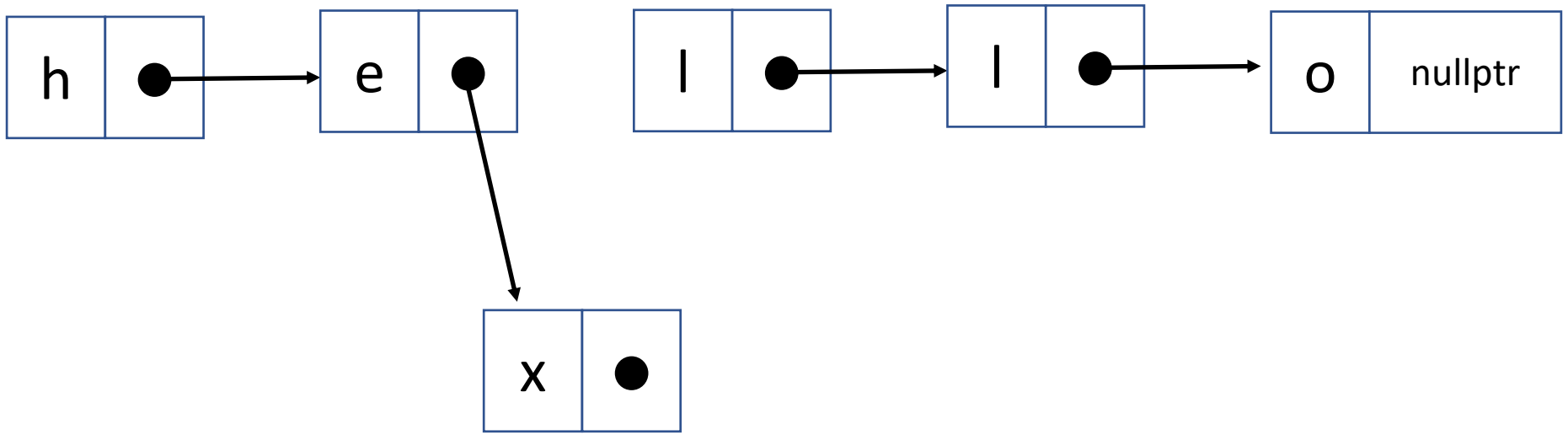
Data una cella, facile inserire una nuova lettera!
Esempio: inseriamo la lettera ‘x’ dopo la ‘e’



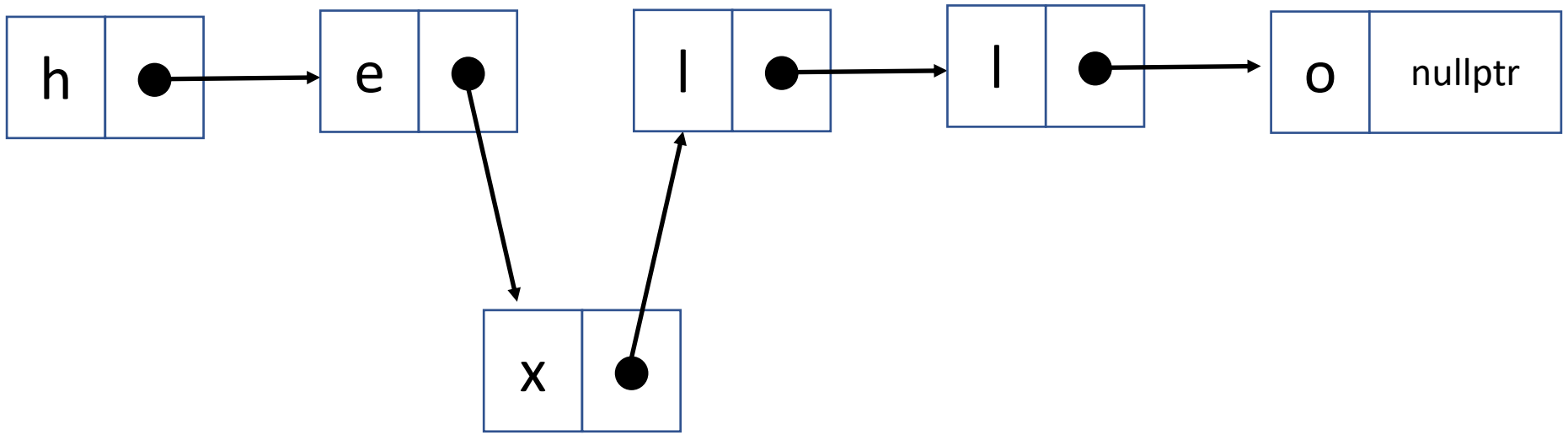
Data una cella, facile inserire una nuova lettera!
Esempio: inseriamo la lettera ‘x’ dopo la ‘e’



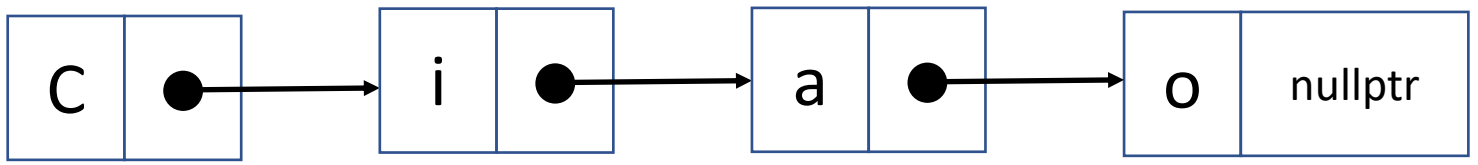
Data una cella, facile inserire una nuova lettera!
Esempio: inseriamo la lettera ‘x’ dopo la ‘e’

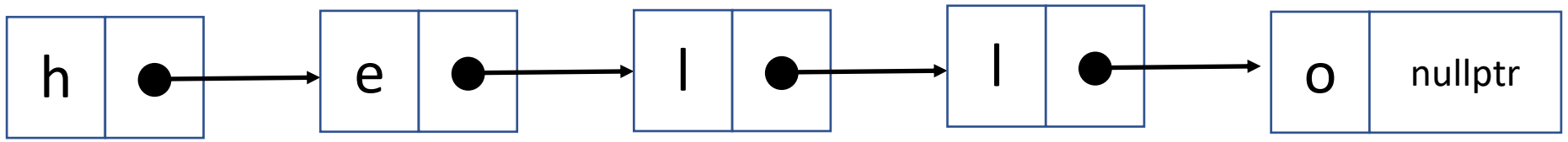


Data una cella, facile inserire una nuova lettera!
Esempio: inseriamo la lettera ‘x’ dopo la ‘e’

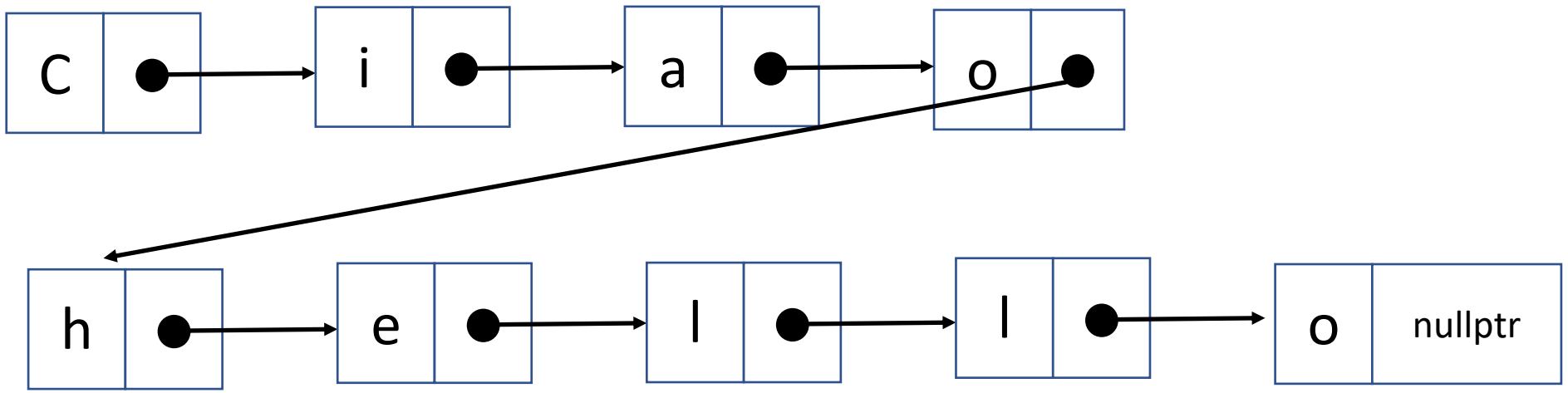


Altra operazione velocissima: concatena due liste
Mi occorrono la prima e ultima cella delle due liste

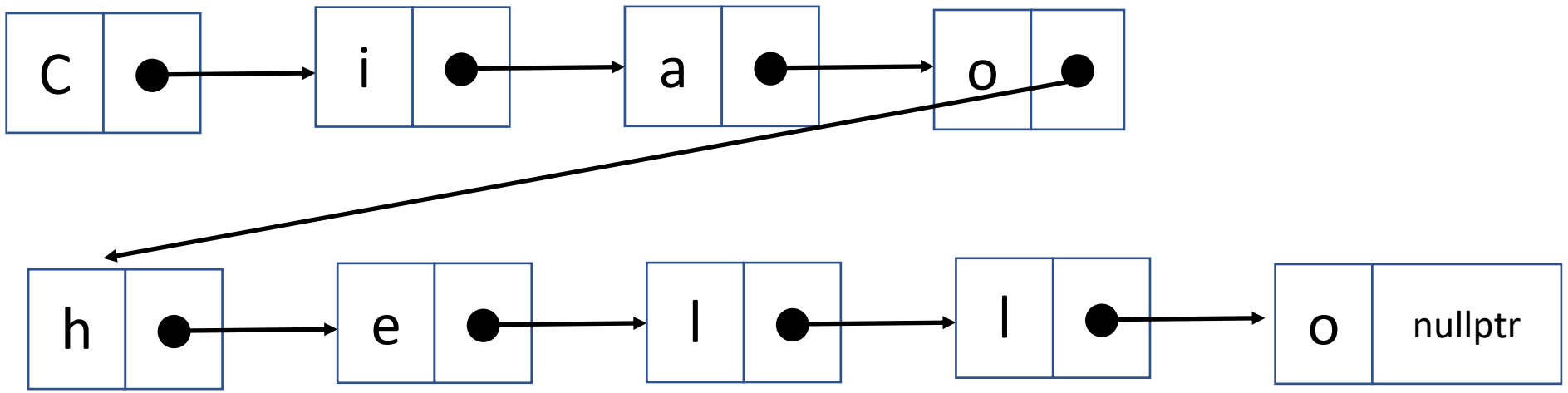




Altra operazione velocissima: concatena due liste
Mi occorrono la prima e ultima cella delle due liste



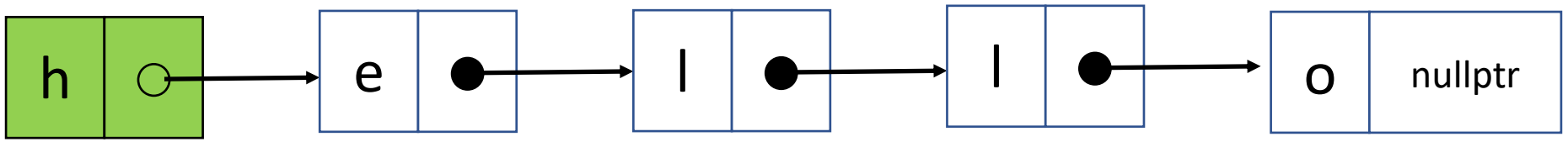
Altra operazione velocissima: concatena due liste
Mi occorrono la prima e ultima cella delle due liste



Nota bene: questa operazione costa O(1), indipendentemente dalla lunghezza delle stringhe (velocissima)

D’altro canto, random access costa parecchio: per accedere alla quarta lettera devo ... partire dall’inizio e contare.

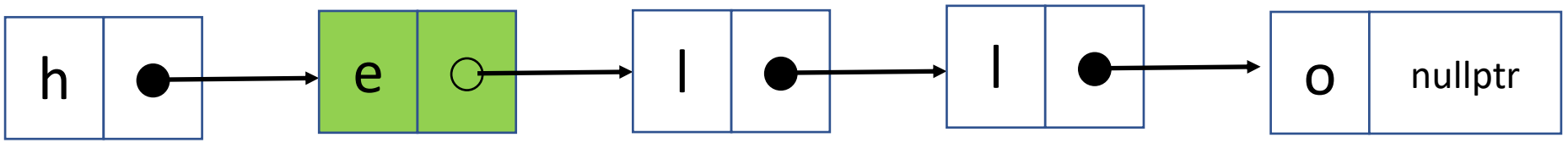
Esempio: estrai la lettera in posizione 3 (contando da 0)



contatore = 0

D’altro canto, random access costa parecchio: per accedere alla quarta lettera devo ... partire dall’inizio e contare.

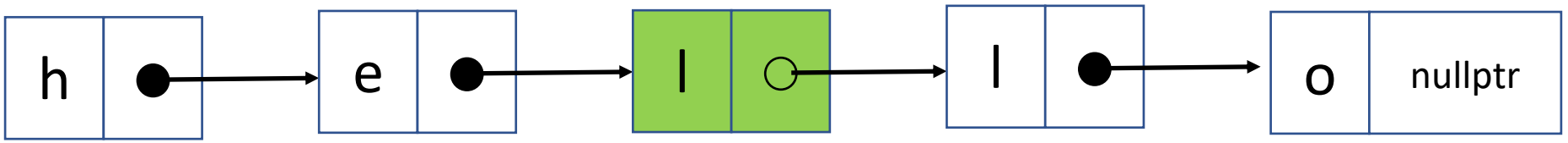
Esempio: estrai la lettera in posizione 3 (contando da 0)



contatore = 1

D’altro canto, random access costa parecchio: per accedere alla quarta lettera devo ... partire dall’inizio e contare.

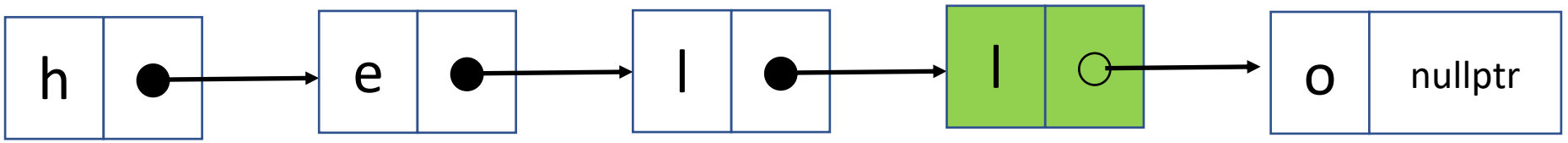
Esempio: estrai la lettera in posizione 3 (contando da 0)



contatore = 2

D’altro canto, random access costa parecchio: per accedere alla quarta lettera devo ... partire dall’inizio e contare.

Esempio: estrai la lettera in posizione 3 (contando da 0)



contatore = 3

Complessità (ottimale / lento)

- **accesso**: O(n)
- **estrai(s):** O(n)
- **concatenazione**: O(1)
- **Inserimenti/cancellazioni/sostituzioni**: O(1) *

* dato un puntatore alla cella da modificare

liste concatenate in C++

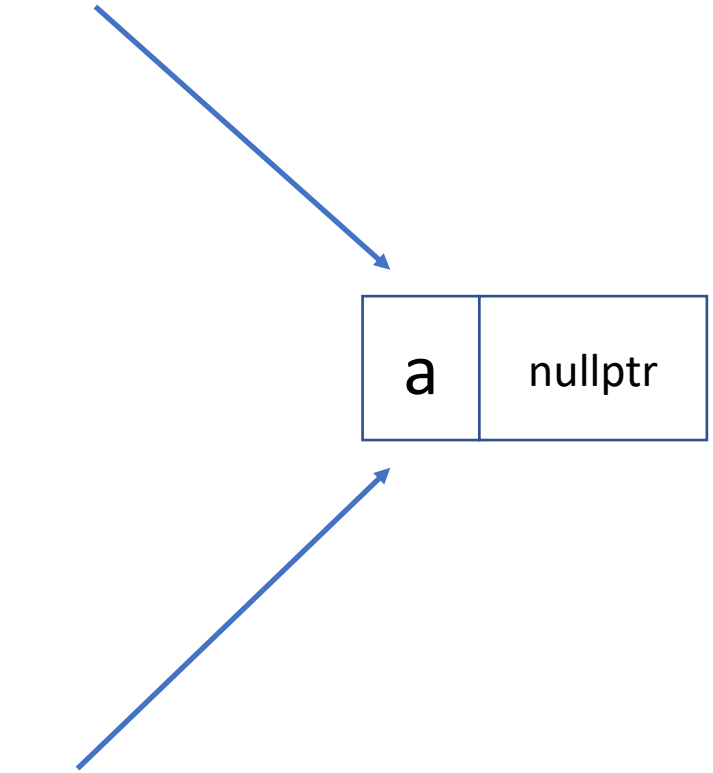
```
struct cella{
        char c;
        cella * next;
}

cella* first = nullptr;
cella* last = nullptr;
```

Una lista è un puntatore alla prima cella e un puntatore all'ultima cella.

Qui creo la lista vuota

Creo una lista con un solo elemento: ‘a’

first = new cella {‘a’, nullptr};



last = first;

Creo una lista con un solo elemento: ‘a’

first = new cella {‘a’, nullptr};

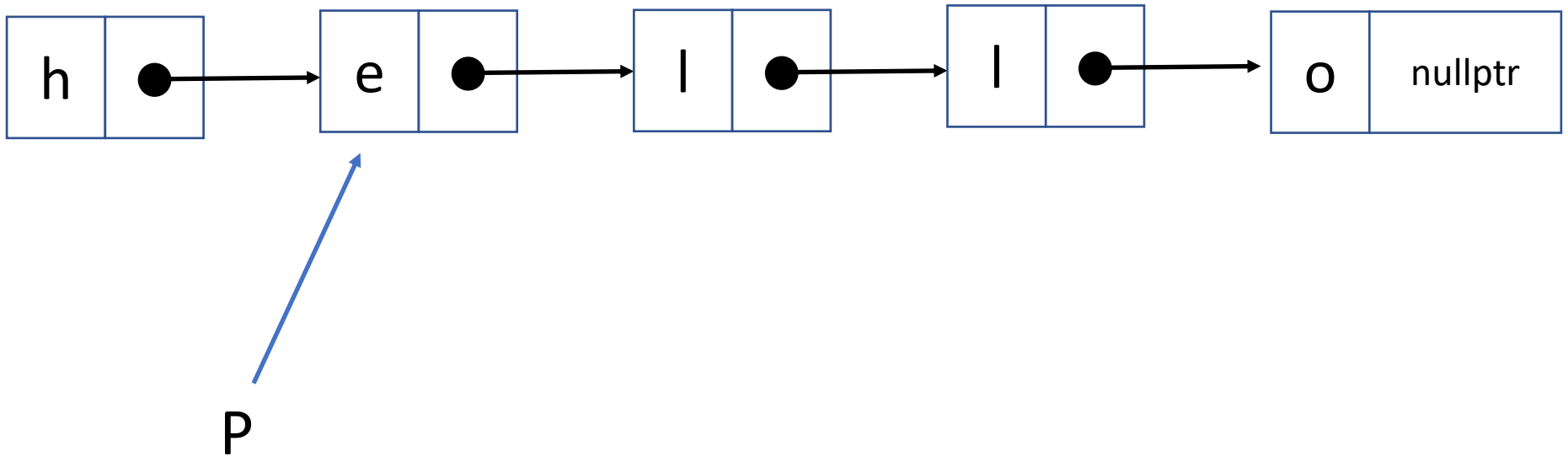
last = first;

crea una cella con contenuto c=‘a’ e next=nullptr

‘first’ e ‘last’ sono puntatori alla cella

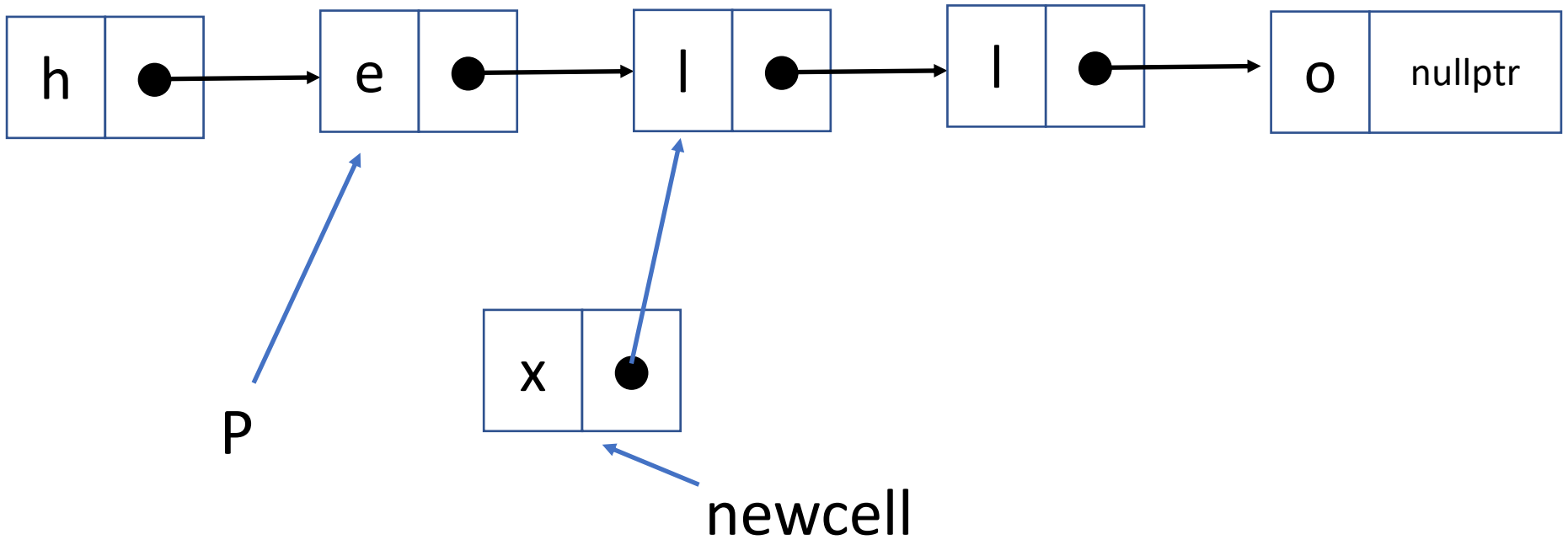
Esempio: inseriamo la lettera ‘x’ dopo la ‘e’

Supponiamo di avere un puntatore P alla cella con la ‘e’



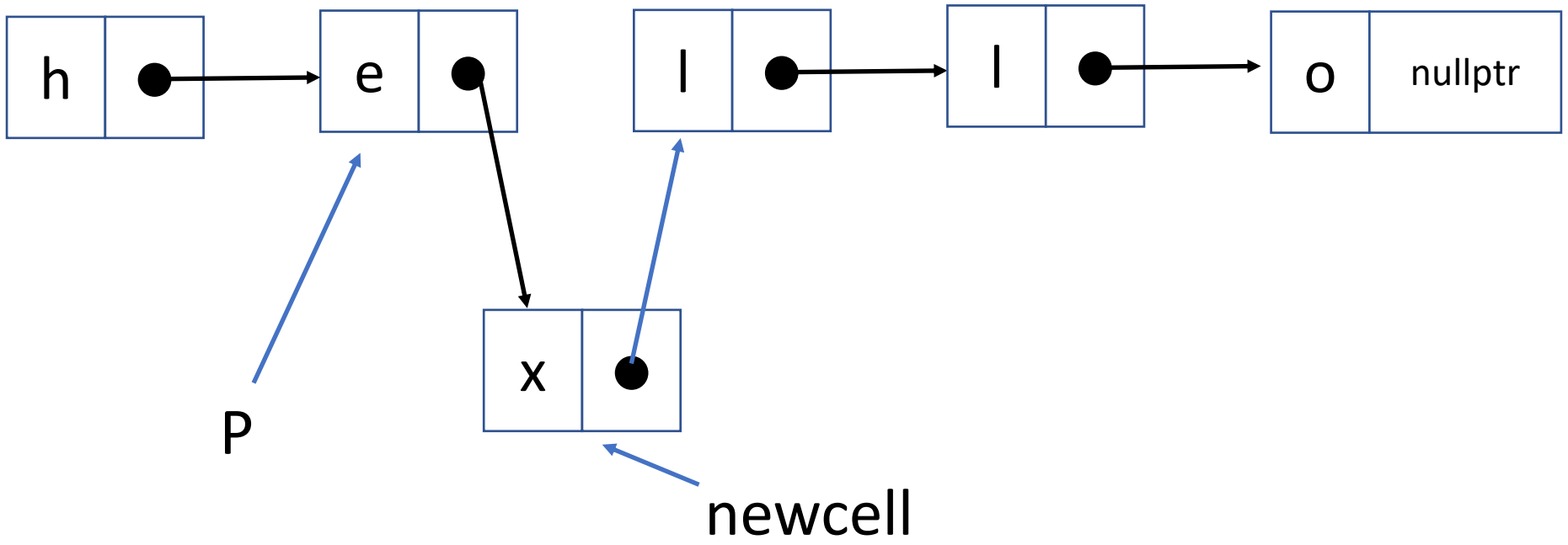
liste concatenate in C++

```
cella* newcell = new cella {‘x’,P->next};
```

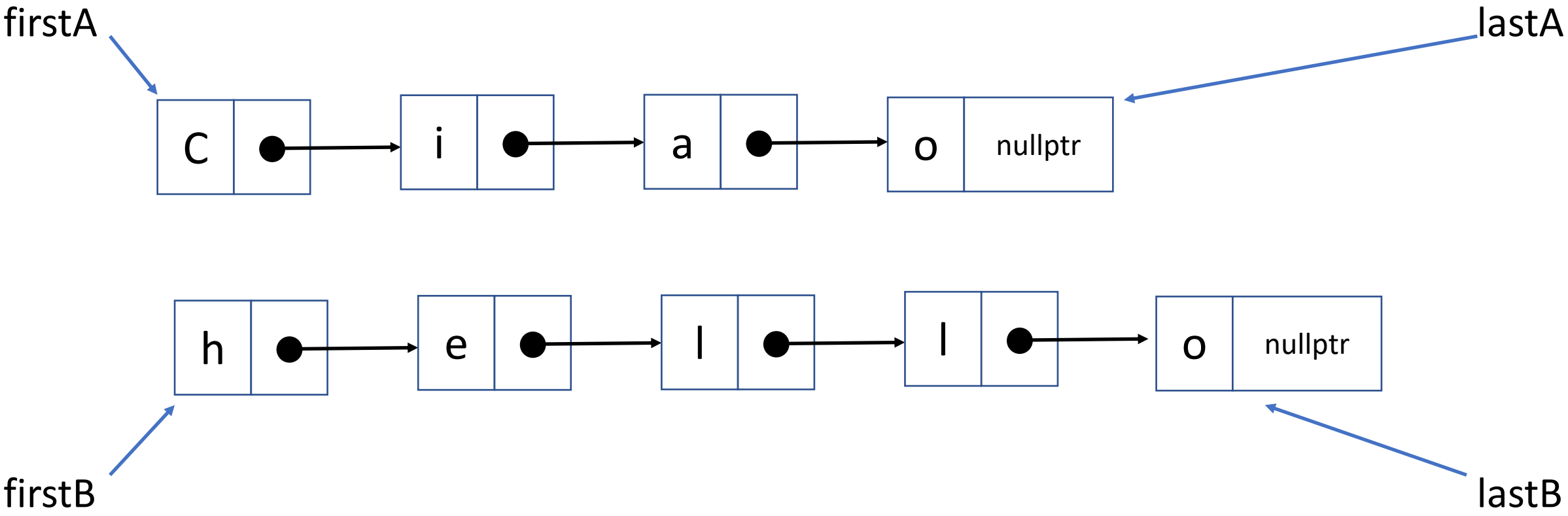


liste concatenate in C++

```
cella* newcell = new cella {‘x’,P->next};
P->next = newcell;
```

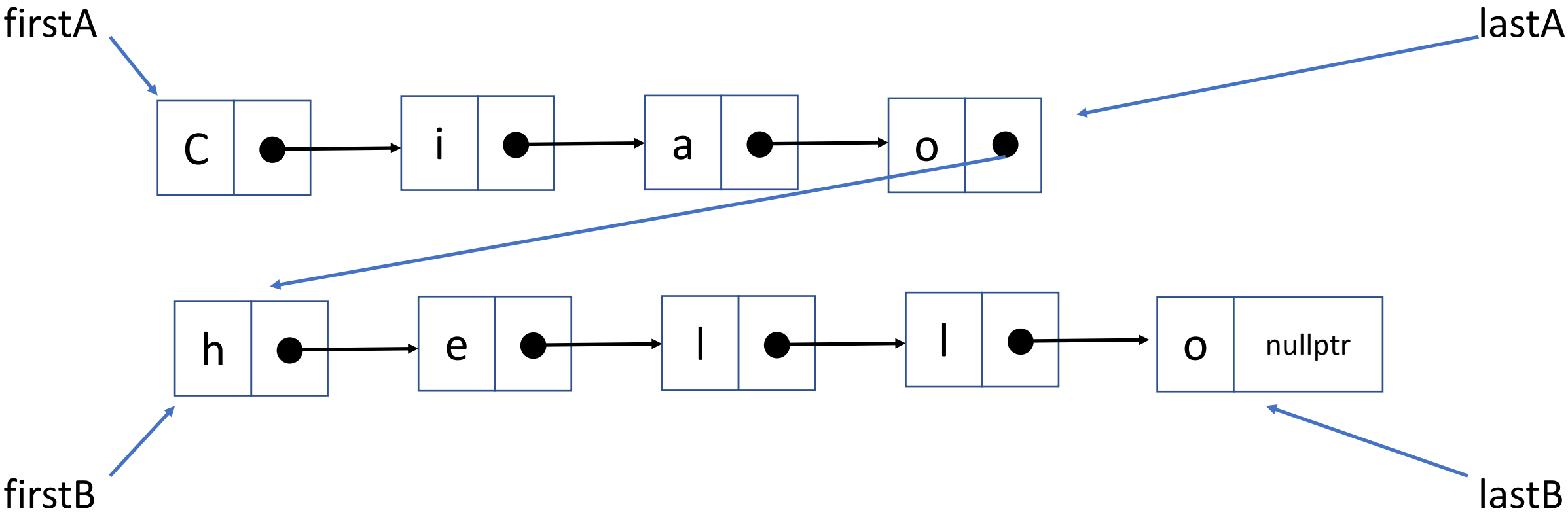


Altro esempio: concatena due liste



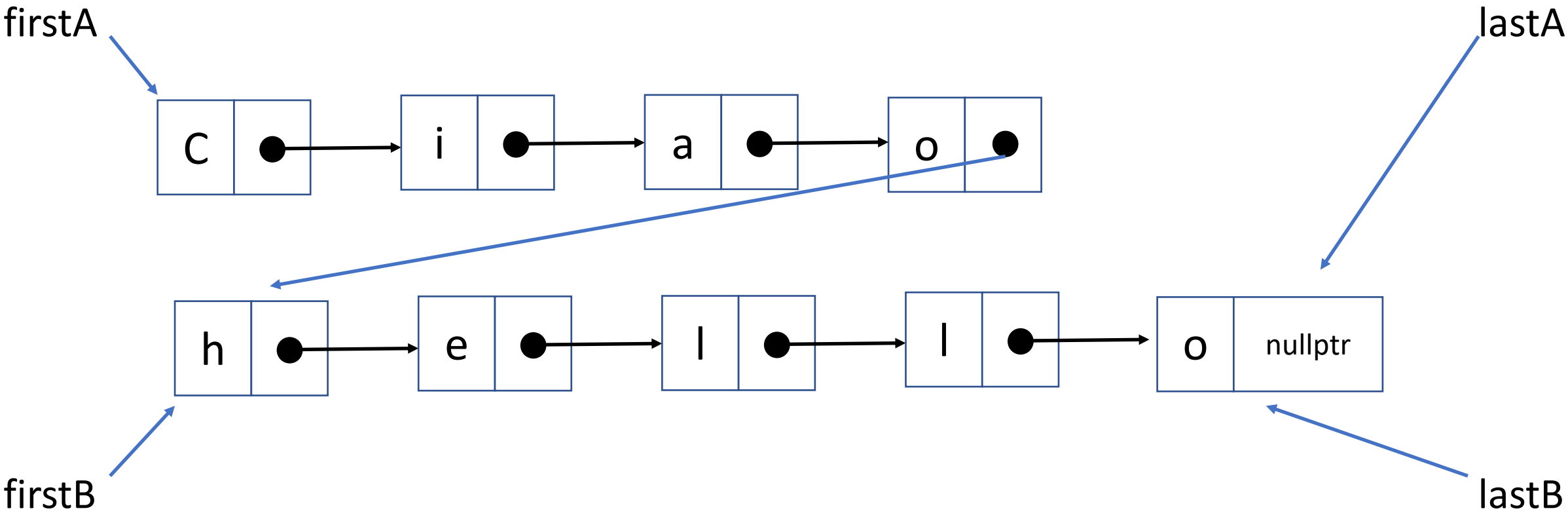
liste concatenate in C++

lastA->next = firstB;



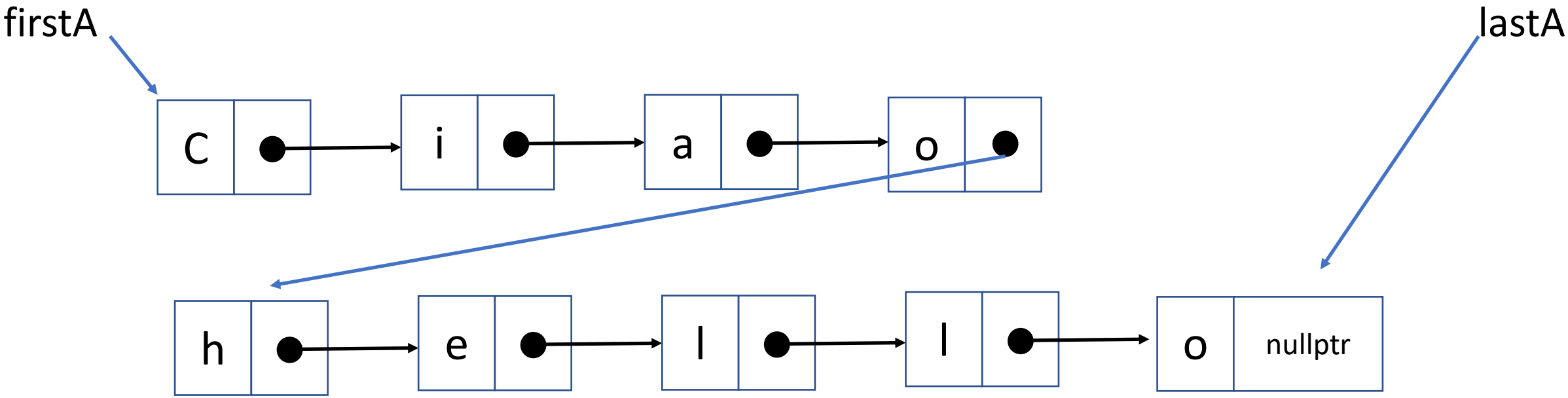
liste concatenate in C++

```
lastA->next = firstB;
lastA = lastB;
```



liste concatenate in C++

```
lastA->next = firstB;
lastA = lastB;
```



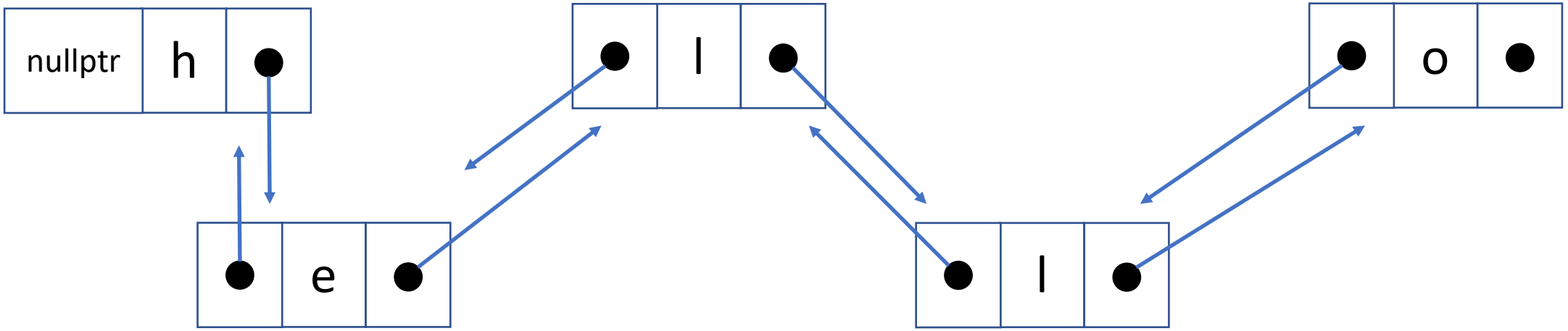
Il risultato della concatenazione è (firstA, lastA)

tempo: O(1)

liste doppiamente concatenate

Qui non le vedremo, ma possiamo anche definire liste **doppiamente concatenate**!

Utili se voglio muovermi avanti/indietro nella stringa



problema: join strings

https://open.kattis.com/problems/joinstrings

Join Strings

You are given a collection of N non-empty strings, denoted by $S_1, S_2, \ldots, S_n$. Then you are given N-1 operations which you execute in the order they are given. The i^{th} operation is has the following format: 'a b' (1-based indexing, without the quotes), which means that you have to make the following changes:

1. $S_a = S_a + S_b$, i.e. concatenate a^{th} string and b^{th} string and store the result in a^{th} string,
2. S_b = "", i.e. make the b^{th} string empty, after doing the previous step.

You are ensured that after the i^{th} operation, there will be no future operation that will be accessing S_b. Given these operations to join strings, print the last string that will remain at the end of this process.

Input

The first line contains an integer N ($1 \leq N \leq 10^5$) denoting the number of strings given. Each of the next N lines contains a string denoting the S_i. All the characters in the string S_i are lowercase alphabets from 'a' to 'z'. The total number of characters over all the strings is at most 10^6, i.e $\sum_{i=1}^{N} |S_i| \leq 10^6$, where $|S_i|$ denotes the length of the i^{th} string. After these N strings, each of the next N-1 lines contain two integers a and b, such that $a \neq b$ and $1 \leq a, b \leq N$ denoting the i^{th} operation.

Join strings

https://open.kattis.com/problems/joinstrings

Sample Input 1

```
4
cute
cat
kattis
is
3 2
4 1
3 4
```

Sample Output 1

```
kattiscatiscute
```

Sample Input 2

```
3
howis
this
practicalexam
1 2
1 3
```

Sample Output 2

```
howisthispracticalexam
```

Join strings

https://open.kattis.com/problems/joinstrings

Ricapitolando, che operazioni veloci su stringhe mi servono?

Join strings

https://open.kattis.com/problems/joinstrings

Ricapitolando, che operazioni veloci su stringhe mi servono?

- Concatenazione
- stampare una stringa intera (l'ultima rimasta)

Quale struttura è la più adatta?

Join strings

https://open.kattis.com/problems/joinstrings

Ricapitolando, che operazioni veloci su stringhe mi servono?

- Concatenazione
- stampare una stringa intera (l'ultima rimasta)

Quale struttura è la più adatta? liste concatenate!

Join strings

https://open.kattis.com/problems/joinstrings

Facciamo un po' di **analisi**. Supponiamo di avere N stringhe, tutte lunghe m (semplifichiamo).

Usando una lista concatenata per ogni stringa:

- Le N concatenazioni mi costano in tutto:

Join strings

https://open.kattis.com/problems/joinstrings

Facciamo un po' di **analisi**. Supponiamo di avere N stringhe, tutte lunghe m (semplifichiamo).

Usando una lista concatenata per ogni stringa:

- Le N concatenazioni mi costano in tutto: O(N)
- Alla fine, stampare la stringa risultante mi costa:

Join strings

https://open.kattis.com/problems/joinstrings

Facciamo un po' di **analisi**. Supponiamo di avere N stringhe, tutte lunghe m (semplifichiamo).

Usando una lista concatenata per ogni stringa:

- Le N concatenazioni mi costano in tutto: O(N)
- Alla fine, stampare la stringa risultante mi costa: O(N*m)

Complessità totale: O(N*m). Ottimale perchè devo comunque stampare a schermo N*m caratteri.

Join strings

https://open.kattis.com/problems/joinstrings

E invece, usando std::string??

- concatenare le N stringhe:

https://open.kattis.com/problems/joinstrings

E invece, usando std::string??

- concatenare le N stringhe, nel caso peggiore: 2m + 3m + 4m + ... + N*m = $O(N^2 * m)$

https://open.kattis.com/problems/joinstrings

E invece, usando std::string??

- concatenare le N stringhe, nel caso peggiore: 2m + 3m + 4m + ... + N*m = $O(N^2 * m)$
- Alla fine, stampare la stringa risultante mi costa: O(N*m)

Totale: $O(N^2 * m)$. Lentissimo!

Join strings

Morale: scegliete bene la struttura dati giusta!

std::string è meglio delle liste se dovete fare tanti random access (s[i])

Strutture per stringhe

troppo complicati per questa lezione!

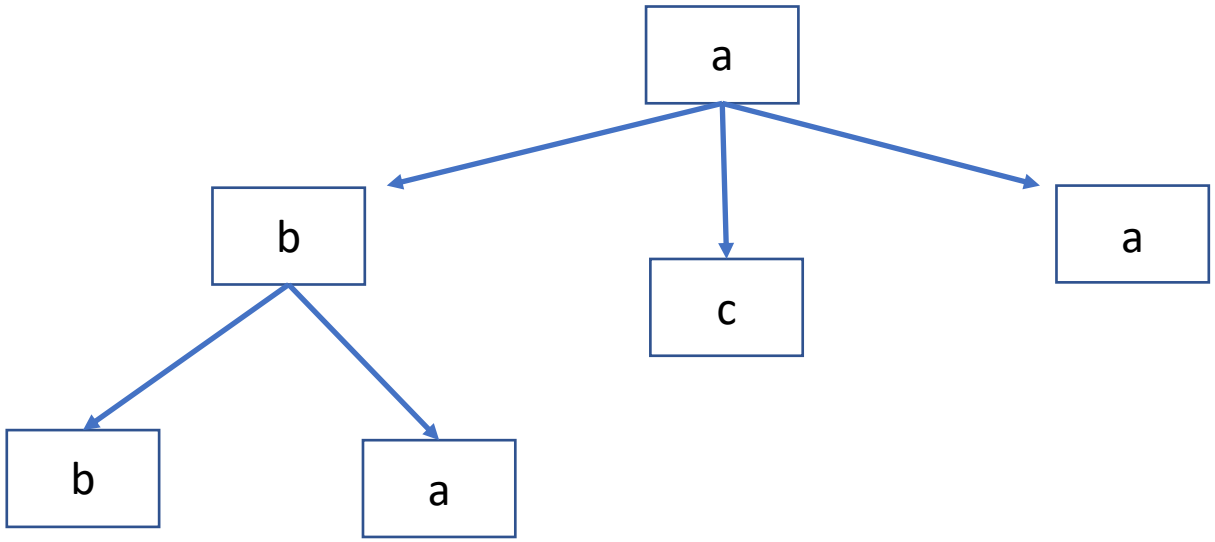
Trade-offs (n = lunghezza stringa)

	lista concatenata	std::string / char[]	alberi bilanciati
random access	O(n)	O(1)	O(log n)
inserimento/cancellazione	O(1)	O(n)	O(log n)
concatenazione	O(1)	O(n)	O(log n)
stampa tutta la stringa	O(n)	O(n)	O(n)
sostituzioni	O(1)	O(1)	O(log n)

tries

Un trie è una struttura adatta a memorizzare un insieme di stringhe

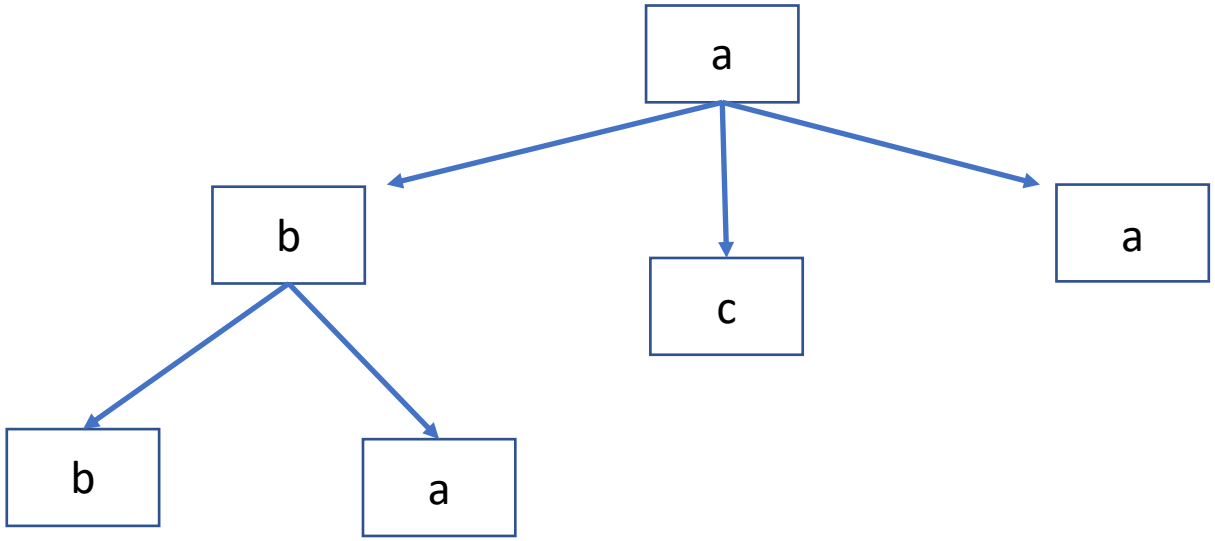
Simile a una lista (puntatori), ma con struttura ad albero (molteplici successori):



Insieme: {abb, aba, ac, aa}

tries in C++

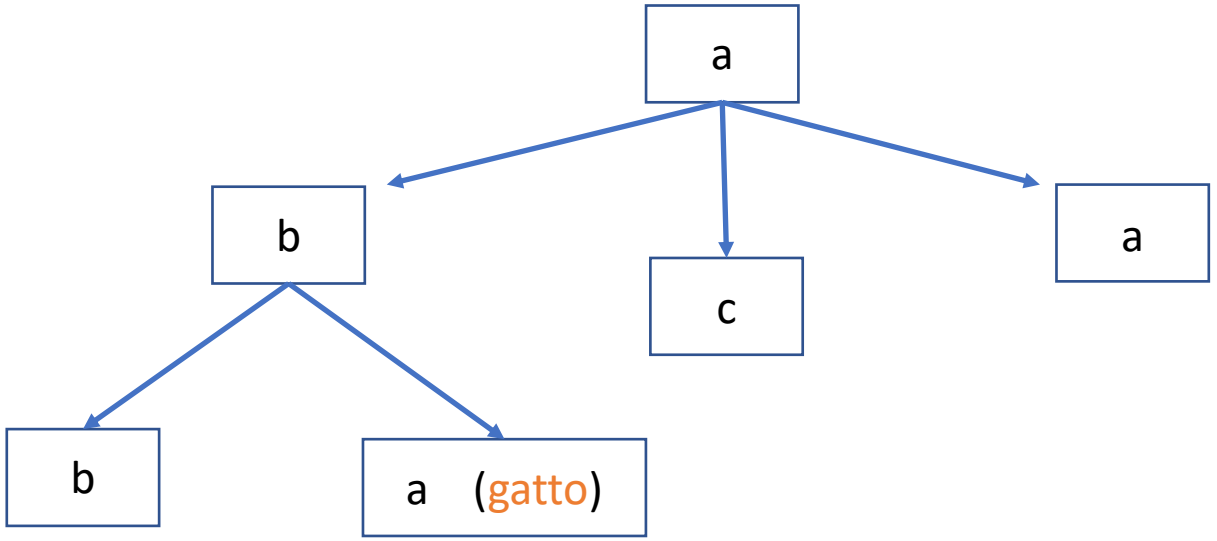
```
struct nodo{
        char c;
        vector<nodo*> puntatori_ai_figli;
        //eventuale informazione aggiuntiva del nodo
}
```



Insieme: {abb, aba, ac, aa}

Quando è utile?

- utile se voglio memorizzare un insieme di stringhe ed associare ad ognuna di esse qualche informazione (esempio: un insieme di codici: aba -> gatto)
- Il trie è veloce da navigare dall'alto verso il basso.



Insieme: {abb, aba, ac, aa}

Esercizi per casa

Problemi su stringhe che richiedono tecniche viste oggi:

- Crittografia https://training.olinfo.it/#/task/crittografia/statement
- Encoded message https://open.kattis.com/problems/encodedmessage

Difficile:

- Stringhe di Fibonacci: https://training.olinfo.it/#/task/fibstr/statement