More on graphs

Binary Lifting, Cycle Detection e Cammini Euleriani

Lorenzo Ferrari, Davide Bartoli

March 8, 2023

Table of contents

Binary Lifting

Problema motivazionale

Definizione: Grafo funzionale

Si dice **grafo funzionale** un grafo in cui ogni nodo esattamente un arco uscente. Gli archi sono nella forma $(v, nxt(v))$ per qualche funzione $nxt : V \to V$

Binary Lifting

Problema motivazionale

Definizione: Grafo funzionale

Si dice **grafo funzionale** un grafo in cui ogni nodo esattamente un arco uscente. Gli archi sono nella forma $(v, nxt(v))$ per qualche funzione $nxt : V \to V$

Binary Lifting

Dato un grafo funzionale G con $N \leq 2 \cdot 10^5$ nodi, rispondi a $Q \leq 2 \cdot 10^5$ query che ti chiedono dove si trova il nodo x dopo $K \leq 10^9$ step.
https://cses.fi/problemset/task/1750/

Binary Lifting

Problema motivazionale

Definizione: Grafo funzionale

Si dice **grafo funzionale** un grafo in cui ogni nodo esattamente un arco uscente. Gli archi sono nella forma $(v, nxt(v))$ per qualche funzione $nxt : V \to V$

Binary Lifting

Dato un grafo funzionale G con $N \leq 2 \cdot 10^5$ nodi, rispondi a $Q \leq 2 \cdot 10^5$ query che ti chiedono dove si trova il nodo x dopo $K \leq 10^9$ step.
https://cses.fi/problemset/task/1750/

- idee?

Binary Lifting

Idea

Un'opzione è visitare uno alla volta i successori di x

[1] questo è *quasi* vero, dobbiamo fare attenzione a K dispari

Binary Lifting

Idea

Un'opzione è visitare uno alla volta i successori di x

- questa soluzione è ovviamente troppo lenta
- la complessità è $O(QK)$

[1] questo è *quasi* vero, dobbiamo fare attenzione a K dispari

Binary Lifting

Idea

Un'opzione è visitare uno alla volta i successori di x

- questa soluzione è ovviamente troppo lenta
- la complessità è $O(QK)$

Possiamo ottimizzare? Supponiamo di costruire un altro grafo funzionale G' con gli archi $(v, nxt(nxt(v)))$. Il $K/2$-esimo successore di x in G' è il K-esimo successore di x in G[1].

[1] questo è *quasi* vero, dobbiamo fare attenzione a K dispari

Binary Lifting

Idea

Un'opzione è visitare uno alla volta i successori di x

- questa soluzione è ovviamente troppo lenta
- la complessità è $O(QK)$

Possiamo ottimizzare? Supponiamo di costruire un altro grafo funzionale G' con gli archi $(v, nxt(nxt(v)))$. Il $K/2$-esimo successore di x in G' è il K-esimo successore di x in G[1].

Così impieghiamo $K/2$ salti per raggiungere il K-esimo successore.

[1] questo è *quasi* vero, dobbiamo fare attenzione a K dispari

Binary Lifting

Idea

Un'opzione è visitare uno alla volta i successori di x

- questa soluzione è ovviamente troppo lenta
- la complessità è $O(QK)$

Possiamo ottimizzare? Supponiamo di costruire un altro grafo funzionale G' con gli archi $(v, nxt(nxt(v)))$. Il $K/2$-esimo successore di x in G' è il K-esimo successore di x in G[1].

Così impieghiamo $K/2$ salti per raggiungere il K-esimo successore.

Notiamo che il problema per G' è analogo al problema originale, continuiamo a usare la stessa ottimizzazione!

[1] questo è *quasi* vero, dobbiamo fare attenzione a K dispari

Binary Lifting

Implementazione

Sia $up[v][j]$ il 2^j-esimo successore di v. $up[v][j]$ indica un salto lungo 2^j.

- $up[v][0] = nxt(v)$
- $up[v][i] = up[up[v][i-1]][i-1] \; \forall i \geq 1$

Binary Lifting

Implementazione

Sia $up[v][j]$ il 2^j-esimo successore di v. $up[v][j]$ indica un salto lungo 2^j.

- $up[v][0] = nxt(v)$
- $up[v][i] = up[up[v][i-1]][i-1] \; \forall i \geq 1$

Per rispondere a una query, consideriamo K nella sua rappresentazione binaria e costruiamo K come composizione di salti lunghi 2^j per $O(\log K)$ valori di j.

- complessità di tempo: $O((Q + N) \log K)$
- complessità di spazio: $O(N \log K)$

Implementazione

```
static constexpr int LOG = 31;
vector<int> up[LOG];

void build(int n, vector<int> p) {
    for (int i = 0; i < LOG; ++i) {
        up[i].resize(n);
    }
    for (int i = 0; i < n; ++i) {
        up[0][i] = p[i];
    }
    for (int j = 1; j < LOG; ++j) {
        for (int i = 0; i < n; ++i) {
            up[j][i] = up[j-1][up[j-1][i]];
        }
    }
}
```

Implementazione

```
int lift(int v, int k) {
    for (int i = 0; i < LOG; ++i) {
        if (k & (1 << i)) {
            v = up[i][v];
        }
    }
    return v;
}
```

Binary Lifting

Lowest Common Ancestor

LCA

Dato un albero radicato con $N \leq 2 \cdot 10^5$ nodi, trova per Q coppie di nodi (a, b) il loro minimo antenato comune.
https://training.olinfo.it/#/task/lca/statement

Binary Lifting

Lowest Common Ancestor

LCA

Dato un albero radicato con $N \leq 2 \cdot 10^5$ nodi, trova per Q coppie di nodi (a, b) il loro minimo antenato comune.
https://training.olinfo.it/#/task/lca/statement

- idee?

LCA

Dato un albero radicato con $N \leq 2 \cdot 10^5$ nodi, trova per Q coppie di nodi (a, b) il loro minimo antenato comune.
https://training.olinfo.it/#/task/lca/statement

- idee?
- **algoritmo naive**
  - alziamo il nodo più profondo fino all'altezza dell'altro
  - finchè a, b non coincidono, $a := nxt(a), b := nxt(b)$
  - a, b si incontrano nell'lca

Binary Lifting

Lowest Common Ancestor

LCA

Dato un albero radicato con $N \leq 2 \cdot 10^5$ nodi, trova per Q coppie di nodi (a, b) il loro minimo antenato comune.
https://training.olinfo.it/#/task/lca/statement

- idee?
- **algoritmo naive**
  - alziamo il nodo più profondo fino all'altezza dell'altro
  - finchè a, b non coincidono, $a := nxt(a), b := nxt(b)$
  - a, b si incontrano nell'lca
- la complessità al momento è $O(n)$ per query

Binary Lifting

Lowest Common Ancestor

Possiamo velocizzare entrambi gli step dell'algoritmo naive con **binary lifting**.

Binary Lifting

Lowest Common Ancestor

Possiamo velocizzare entrambi gli step dell'algoritmo naive con **binary lifting**.

- calcoliamo la profondità $dep[v]$ di ogni nodo
- (wlog) il nodo b è più profondo del nodo a
- $b := lift(b, dep[b] - dep[a])$
- se $a = b$, l'lca è a
- altrimenti raggiungiamo in $\log N$ salti i nodi a', b' appena sotto l'lca
  - (approfondiamo nell'implementazione)

Binary Lifting

Lowest Common Ancestor

Possiamo velocizzare entrambi gli step dell'algoritmo naive con **binary lifting**.

- calcoliamo la profondità $dep[v]$ di ogni nodo
- (wlog) il nodo b è più profondo del nodo a
- $b := lift(b, dep[b] - dep[a])$
- se $a = b$, l'lca è a
- altrimenti raggiungiamo in $\log N$ salti i nodi a', b' appena sotto l'lca
  - (approfondiamo nell'implementazione)

Complessità di tempo: $O(\log N)$ per query

LCA

Implementazione

```cpp
int lca(int a, int b) {
    if (dep[a] > dep[b]) swap(a, b);
    b = lift(b, dep[b] - dep[a]);
    if (a == b) return a;
    // IMPORTANTE: le potenze di 2 dalla più grande
    for (int i = LOG-1; i >= 0; --i) {
        if (up[i][a] != up[i][b]) {
            a = up[i][a];
            b = up[i][b];
        }
    }
    assert(up[0][a] == up[0][b]);
    return up[0][a];
}
```

LCA

Varianti

Analogamente al segment, anche la **Sparse Table** per binary lifting è molto versatile.

Analogamente al segment, anche la **Sparse Table** per binary lifting è molto versatile.

Purché non ci siano update.

Analogamente al segment, anche la **Sparse Table** per binary lifting è molto versatile.

Purché non ci siano update.

In generale, possiamo usare una generica struct `nodo` calcolabile da due nodi figli. Per esempio, possiamo rispondere a query di minimo/massimo/somma di un percorso, subarray con somma massima ecc.

https://training.olinfo.it/#/task/lca/statement

Cycle Detection

Problema

Cycle Detection 1

Dato un grafo diretto con $N \leq 2 \cdot 10^5$ nodi e $M \leq 5 \cdot 10^5$ archi, stampare, se esiste, un ciclo.

Cycle Detection

Problema

Cycle Detection 1

Dato un grafo diretto con $N \leq 2 \cdot 10^5$ nodi e $M \leq 5 \cdot 10^5$ archi, stampare, se esiste, un ciclo.

- idee?

Cycle Detection

Problema

Cycle Detection 1

Dato un grafo diretto con $N \leq 2 \cdot 10^5$ nodi e $M \leq 5 \cdot 10^5$ archi, stampare, se esiste, un ciclo.

- idee?
- pensiamo a come funziona la DFS. In ogni momento i nodi possono essere:
  1. attivi
  2. non attivi e non ancora visitati
  3. non attivi e già visitati

Osservazione chiave

Se in un qualsiasi momento uno dei nostri vicini è un nodo attivo, allora siamo in un ciclo.

Cycle Detection 2

Dato un grafo pesato con $N \leq 2500$ nodi e $M \leq 5000$ archi, dire se esiste un ciclo con peso negativo.
https://cses.fi/problemset/task/1197/

Cycle Detection

Problema

Cycle Detection 2

Dato un grafo pesato con $N \leq 2500$ nodi e $M \leq 5000$ archi, dire se esiste un ciclo con peso negativo.
https://cses.fi/problemset/task/1197/

- ricordiamo che l'algoritmo di Dijkstra non termina se il grafo contiene un ciclo negativo.
- soluzione "sbagliata": facciamo Dijkstra, se è troppo lento esiste un ciclo negativo e lo fermiamo

Cycle Detection

Problema

Cycle Detection 2

Dato un grafo pesato con $N \leq 2500$ nodi e $M \leq 5000$ archi, dire se esiste un ciclo con peso negativo.
`https://cses.fi/problemset/task/1197/`

- ricordiamo che l'algoritmo di Dijkstra non termina se il grafo contiene un ciclo negativo.
- soluzione "sbagliata": facciamo Dijkstra, se è troppo lento esiste un ciclo negativo e lo fermiamo
- soluzione legit: controlliamo se l'algoritmo di **Bellman-ford** va oltre la $(n-1)$-esima iterazione

Cammini Euleriani

Problema

Cammino Euleriano

Dato un grafo non diretto, trova, se esite, un cammino euleriano, ovvero un cammino che passa per ogni arco esattamente una volta.
`https://training.olinfo.it/#/task/oii_matita/statement`

Cammini Euleriani

Problema

Cammino Euleriano

Dato un grafo non diretto, trova, se esite, un cammino euleriano, ovvero un cammino che passa per ogni arco esattamente una volta.
https://training.olinfo.it/#/task/oii_matita/statement

Come prima cosa cerchiamo di capire quando un cammino euleriano esiste.

Cammino Euleriano

Dato un grafo non diretto, trova, se esite, un cammino euleriano, ovvero un cammino che passa per ogni arco esattamente una volta.
https://training.olinfo.it/#/task/oii_matita/statement

Come prima cosa cerchiamo di capire quando un cammino euleriano esiste.

Consideriamo un singolo nodo. Se il numero di archi incidenti è pari, allora il numero di volte che "entriamo" nel nodo è uguale al numero di volte che "usciamo" dal nodo, quindi questo nodo può essere "nel mezzo" del nostro cammino.

Cammini Euleriani

Problema

Cammino Euleriano

Dato un grafo non diretto, trova, se esite, un cammino euleriano, ovvero un cammino che passa per ogni arco esattamente una volta.
https://training.olinfo.it/#/task/oii_matita/statement

Come prima cosa cerchiamo di capire quando un cammino euleriano esiste.

Consideriamo un singolo nodo. Se il numero di archi incidenti è pari, allora il numero di volte che "entriamo" nel nodo è uguale al numero di volte che "usciamo" dal nodo, quindi questo nodo può essere "nel mezzo" del nostro cammino.
Se invece il numero di archi incidenti è dispari, allora il nodo deve essere per forza il nodo da cui partiamo o il nodo in cui finiamo.

Cammini Euleriani

Ora quindi sappiamo controllare se il cammino esiste: se il numero di nodi con grado dispari è 0 o 2, allora il cammino esiste, altrimenti no.

Cammini Euleriani

Ora quindi sappiamo controllare se il cammino esiste: se il numero di nodi con grado dispari è 0 o 2, allora il cammino esiste, altrimenti no.

Come facciamo a trovare il cammino però?

Cammini Euleriani

Ora quindi sappiamo controllare se il cammino esiste: se il numero di nodi con grado dispari è 0 o 2, allora il cammino esiste, altrimenti no.

Come facciamo a trovare il cammino però?

In realtà è semplice, basta fare una dfs mantenedo l'array dei visitati sugli archi invece che sui nodi.

Cammini Euleriani

Implementazione

```cpp
// nodo di destinazione, indice arco
vector<pair<int, int> > adj[100010];
void dfs(int p) {
    for (int i = 0; i < adj[p].size(); i++) {
        if (vis[adj[p][i].second] == 1) continue;
        vis[adj[p][i].second] = 1;
        dfs(adj[p][i].first);
    }
    sol.push_back(p);
}
// se ci sono 2 nodi con grado dispari, allora
// devo chiamare  dfs su uno di questi nodi
```

Cammini Euleriani

Vedere perché il codice funziona non è ovvio, bisogna ragionarci un pochino.

Cammini Euleriani

Vedere perché il codice funziona non è ovvio, bisogna ragionarci un pochino.
Possiamo fare alcune osservazioni:

- costruiamo il cammino in ordine inverso

Cammini Euleriani

Vedere perché il codice funziona non è ovvio, bisogna ragionarci un pochino.
Possiamo fare alcune osservazioni:

- costruiamo il cammino in ordine inverso
- se troviamo un cammino fino al nodo finale, quello che succede è che mentre "torniamo indietro" aggiungiamo dei cicli che passano per il nodo corrente

Cammini Euleriani

Vedere perché il codice funziona non è ovvio, bisogna ragionarci un pochino.
Possiamo fare alcune osservazioni:

- costruiamo il cammino in ordine inverso
- se troviamo un cammino fino al nodo finale, quello che succede è che mentre "torniamo indietro" aggiungiamo dei cicli che passano per il nodo corrente
- utilizziamo sempre tutti gli archi (se rispetta le condizioni del cammino euleriano)

Cammini Euleriani

Vedere perché il codice funziona non è ovvio, bisogna ragionarci un pochino.
Possiamo fare alcune osservazioni:

- costruiamo il cammino in ordine inverso
- se troviamo un cammino fino al nodo finale, quello che succede è che mentre "torniamo indietro" aggiungiamo dei cicli che passano per il nodo corrente
- utilizziamo sempre tutti gli archi (se rispetta le condizioni del cammino euleriano)

Problemi

https://cses.fi/problemset/task/1750/
https://cses.fi/problemset/task/1160/
https://cses.fi/problemset/task/1197/
https://training.olinfo.it/#/task/oii_matita/statement
https://training.olinfo.it/#/task/nostar/statement
https://training.olinfo.it/#/task/lca/statement
https://training.olinfo.it/#/task/itoi_vsmovies/statement