ASD Laboratorio 03

Cristian Consonni/Alessio Guerrieri

UniTN

14/11/2017

Calendario

10/10	Introduzione
31/10	Ad-Hoc
14/11	Grafi 1
28/11	Grafi 2
05/12	Progetto 1
12/12	Progetto 1

Progetto:

- 05 – 13 dicembre;
- Iscrizione dei gruppi ai progetti entro il **04 dicembre**:

  `http://bit.ly/ASDprog`

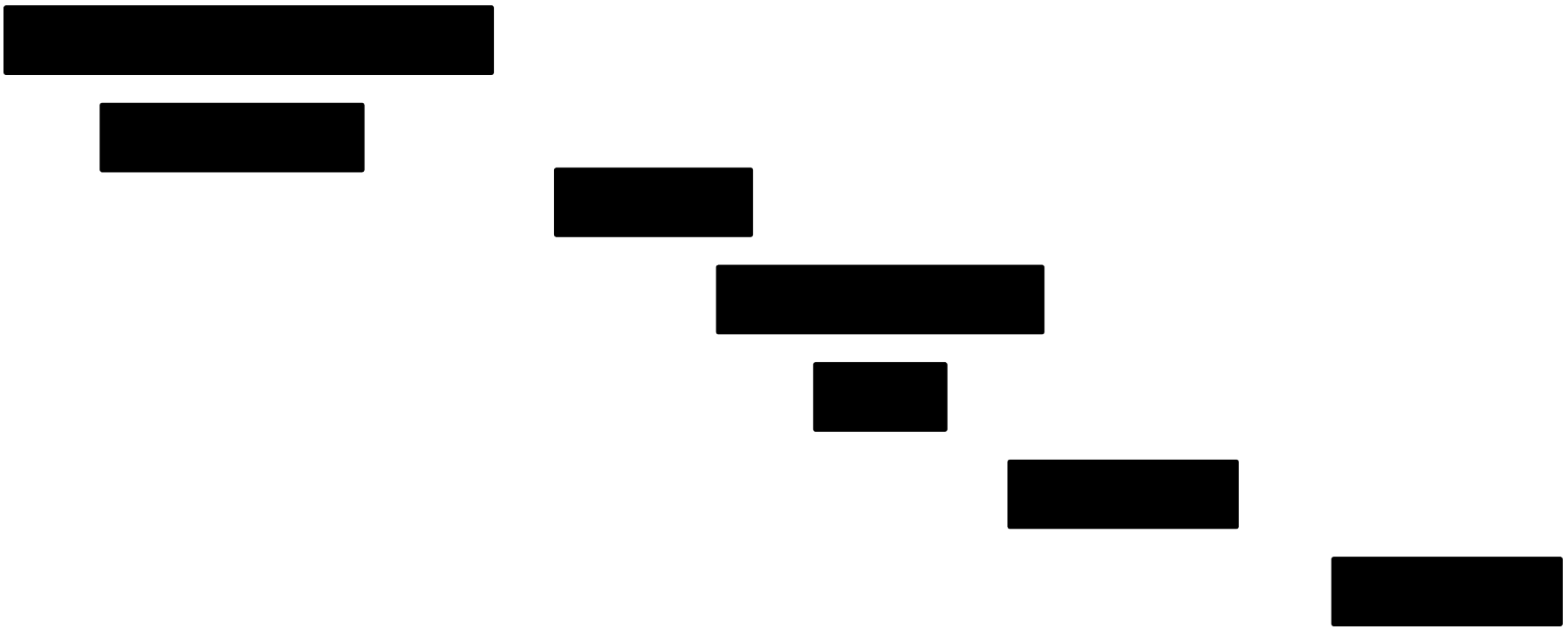
MERGE SORT

```
vector<int> merge_sort(vector<int>& arr){
  if(arr.size()<=1)
    return arr;

  //Spezzo l'array in due parti
  int m=arr.size()/2;

  vector<int> left(&arr[0],&arr[m]);
  vector<int> right(&arr[m],&arr[arr.size()]);

  //Ordino le due parti
  left=merge_sort(left);
  right=merge_sort(right);
```

```
  // Inserisco guardie
  left.push_back(INT_MAX);
  right.push_back(INT_MAX);

  vector<int> result;
  int l=0,r=0;
  for(int i=0;i<arr.size();i++){
    if(right[r]<left[l]){
      result.push_back(right[r]);
      r++;
    } else {
      result.push_back(left[l]);
      l++;
    }
  }
  return result;
}
```

Intervallo

Dati N intervalli $\{(Inizio_i, Fine_i)\}_{i=1}^{N}$, vogliamo sapere quale è il piú lungo periodo non coperto da alcun intervallo, considerando solo gli istanti compresi fra il minimo istante di inizio ed il massimo istante di fine degli intervalli.



Soluzioni: Intervallo

Simulazione semplice

Se gli intervalli arrivano fino all'istante M, creare un array A[1..M] di booleani.
Per ogni intervallo, segnare gli istanti coperti dall'intervallo.
Visitare l'array per trovare i periodi scoperti.

Simulazione efficente

Prendere tutti gli istanti di inizio e fine intervallo ed ordinarli tutti insieme, mantenendo in memoria quali si riferiscono ad un inizio intervallo e quali ad una fine intervallo.
Analizzare questi istanti mantenendo un contatore con il numero di intervalli attivi.
Se il contatore è uguale a 0, abbiamo trovato un buco.

Soluzioni: Intervallo

1. Ordiniamo gli intervalli per istante di inizio
2. L'intervallo i termina un periodo scoperto se $start[i] > end[j]$ per ogni $j < i$
3. Visitiamo gli intervalli in ordine mantenendo in memoria il massimo degli istanti di fine.

$M \leftarrow end[1]$
for $i = 2 \rightarrow N$ **do**
    if $start[i] > M$ **then**
        "Trovato periodo scoperto da M a $start[i]$"
    end if
    $M \leftarrow max(M, end[i])$
end for

Sort Pesato

Vi viene dato un array di N interi da ordinare. Gli elementi sono precisamente tutti gli interi fra 1 e N. Ad ogni turno potete scambiare due elementi a scelta dell'array. Per fare ciò, pagate un prezzo pari alla somma dei due elementi.

Esempio

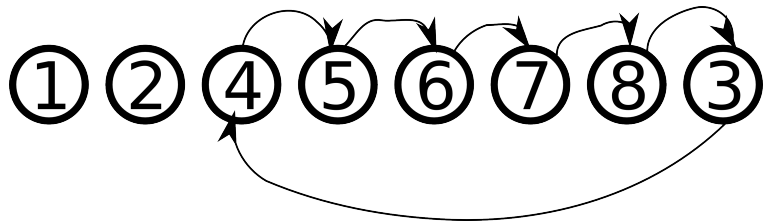
Per scambiare di posto l'elemento 3 e l'elemento 4 impiegate un turno e pagate 7.

Dovete risolvere due problemi: qual è il metodo piú veloce (che ottimizza il numero di turni) e il metodo piú economico (che ottimizza il prezzo).

Soluzione Sort Pesato

- Ogni elemento deve finire in una certa posizione
- Possiamo vederli come archi in un grafo da N nodi e N archi
- Individuiamo i cicli e valutiamo separatamente.
  - Il numero minimo di mosse per un ciclo di N elementi è $N - 1$
  - La soluzione ovvia sposta sempre l'elemento piú piccolo con gli altri elementi
  - Puó convenire "prendere in prestito" l'elemento 1, scambiandolo con l'elemento piú piccolo del ciclo. ordinare il resto del ciclo e rimettere a posto l'elemento 1.

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



$$(3+8)+(3+7)+(3+6)+(3+5)+(3+4)=45$$

$$(1+3)+(1+8)+(1+7)+(1+6)+(1+5)+(1+4)+(3+1)=43$$

Graphs 101

Metodi principali per implementare i grafi

1. Liste di adiacenza (per ogni nodo, lista vicini)
2. Matrice di adiacenza
3. Lista di archi

Esempi in zip sul sito.

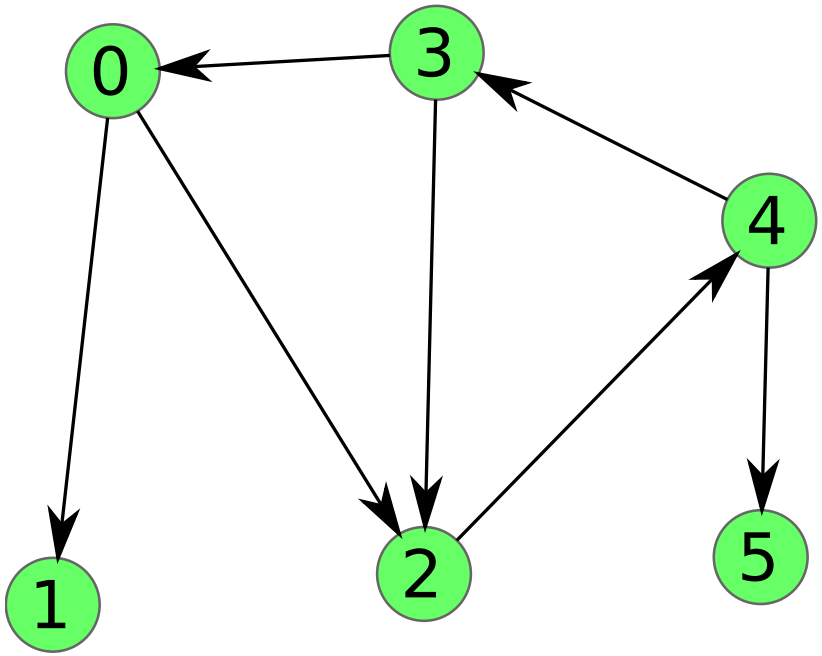
Matrice di adiacenza

Implementazione ovvia con matrice $N \times N$

Formato di input

- Prima riga contiene *N* e *M*, numero di nodi e numero di archi
- Successive *M* righe contengono gli archi (eventualmente con peso)

```
6 7
0 1
0 2
4 3
4 5
3 2
2 4
3 0
```



LISTE DI ADIACENZA

ARRAY DI VECTOR

Lista di adiacenza implementata come array di vector o vector di vector. Se servono altre variabili (open, close) si creano altri array

```
vector< vector <int> > adj;
vector<bool> visitato;
```

ARRAY DI STRUCT

Struct nodo contenente lista di adiacenza e altre informazioni.

```
struct nodo{
 vector<int> vic;
 bool visitato= false;
};
```

LISTE DI ADIACENZA

```
struct nodo{
  vector<int> vic;
  bool visitato= false;
};
vector<nodo> grafo;
...
in >> N;
grafo.resize(N);
for(int i=0;i<M;i++){
  int from, to;
  in >> from >> to;
  grafo[from].vic.push_back(to);
}
```

Problemi

Visita di grafo orientato

Dato un grafo ed un nodo di partenza, trovare il numero di nodi raggiungibili da quel nodo

Diametro su grafo non orientato

Dato un grafo non orientato trovare il piú lungo percorso minimo fra due nodi.

Numero di cammini minimi

Dato un grafo orientato e due nodi, trovare il numero di diversi cammini minimi fra i due nodi.

Problemi

Luddisti spaziali

Primo progetto a. a. 2011/2012

Pokémon

Primo progetto a. a. 2016/2017