

Euristico di tipo list-based scheduling (Progetto 11 a.a. 2019/20)

In questo progetto si realizza il list based scheduling usando come funzione di priorità la lunghezza del critical path (numero massimo di nodi presenti tra un nodo all'uscita) ed usando un numero fissato di componenti, inseriti ad ogni esecuzione del programma. Oltre al file di uscita output.txt contenente lo scheduling finale a latenza minima viene anche prodotto il file log_output.txt che fornisce una traccia dettagliata delle operazioni svolte. Nel caso di errori dovuti agli input errati dell'utente il programma si arresta, lasciando informazioni sul tipo di errore che ha riscontrato. "MainProgetto.java" è il file principale ed utilizza "Nodo.java" per creare oggetti di tipo nodo e per funzionare.

Inizialmente il software legge un file di input di tipo testo (input.txt) contenente le operazioni da schedulare e le inserisce in una struttura a foresta, costituita da Nodi.

Un oggetto di tipo Nodo ha i seguenti attributi:

nome: indica il nome del nodo;

operazione: indica il tipo di operazione;

partenze: lista che mi serve per tenere traccia di quali nodi necessitano che questa operazione sia svolta per poter essere schedulati;

arrivi: lista che mi serve per tenere traccia di quali nodi ho bisogno che siano svolti prima di poter schedulare questa operazione;

radice: serve per identificare il nodo radice, cioè l'operazione che deve essere schedulata per ultima;

foglia: serve per identificare se il nodo è foglia, cioè se l'operazione è pronta per essere schedulata;

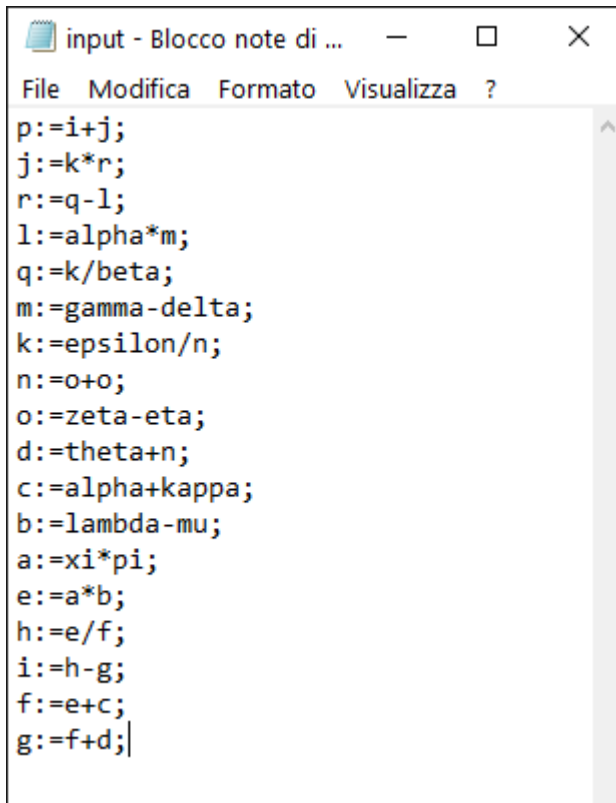
cp: mi serve per memorizzare il critical path;

schedulato: mi serve per memorizzare se il nodo è stato schedulato;

clock: indica in quale ciclo di clock è stato schedulato;

```
7     private String nome;  
8     private String operazione;  
9     private Vector<Nodo> partenze;  
10    private Vector<Nodo> arrivi;  
11    private boolean radice;  
12    private boolean foglia;  
13    private int cp;  
14    private boolean schedulato;  
15    private int clock;
```

Un esempio di file di input (che le ho mandato per e-mail insieme ai sorgenti):



```
input - Blocco note di ...
File Modifica Formato Visualizza ?
p:=i+j;
j:=k*r;
r:=q-l;
l:=alpha*m;
q:=k/beta;
m:=gamma-delta;
k:=epsilon/n;
n:=o+o;
o:=zeta-eta;
d:=theta+n;
c:=alpha+kappa;
b:=lambda-mu;
a:=xi*pi;
e:=a*b;
h:=e/f;
i:=h-g;
f:=e+c;
g:=f+d;
```

Le operazioni disponibili sono {+, -, *, /} e considero le operazioni + e - eseguibili dallo stesso componente.

La parte a sinistra di := rappresenta il nome univoco del nodo. Ogni riga del file di testo corrisponde ad un'operazione e quindi ad un nodo che viene poi inserito in una lista denominata **lista_nodi**. Ho notato che i membri della parte destra sono rilevanti per lo scheduling solo se appartengono alla lista dei nodi. Utilizzo questa informazione per creare i collegamenti tra i nodi in modo opportuno creando quindi la foresta.

Frammento di codice che mostra la creazione dei collegamenti tra due nodi:

```
183         if (appartieneAiNodi(lista_nodi, uno) == true) {
184             for (int j = 0; j < lista_nodi.size(); j++) {
185                 if (lista_nodi.get(j).getNome().equals(uno) == true) {
186                     lista_nodi.get(i).getArrivi().add(lista_nodi.get(j));
187                     lista_nodi.get(j).getPartenze().add(lista_nodi.get(i));
188                     log.write(uno + " appartiene alla lista dei nodi, quindi mi interessa per lo scheduling\n"
189                             + "Aggiungo negli arrivi del nodo " + lista_nodi.get(i).getNome() + " il nodo "
190                             + lista_nodi.get(j).getNome() + " e aggiungo nelle partenze del nodo "
191                             + lista_nodi.get(j).getNome() + " il nodo " + lista_nodi.get(i).getNome() + "\n");
192                 }
193             }
194         }
```

A questo punto abbiamo una lista di nodi collegati fra loro, ed ogni nodo ha tutte le informazioni che caratterizzano l'operazione che rappresenta. Creo un clone di **lista_nodi** denominato **copia** perché durante lo scheduling rimuoverò i nodi che sono stati schedulati con successo.

Lo scheduling è stato realizzato utilizzando un do-while, ogni ciclo corrisponde ad un ciclo di clock ed una volta usciti da questo do-while siamo sicuri che tutte le operazioni sono state schedulate.

Finché ci sono nodi da schedulare vengono eseguite le seguenti operazioni:

- Incremento il numero di clock e ripristino le variabili ausiliarie che rappresentano i componenti a mia disposizione
- Cerco le foglie e le ordino in base al loro critical path
- Finché ho risorse disponibili mando le foglie in scheduling in base alla loro priorità
- Rimuovo i nodi foglia che sono appena stati schedulati dalla lista **copia** e aggiorni i nodi originali in **lista_nodi**

La lista **copia** è stata distrutta ma recupero le informazioni relative allo scheduling dalla lista originale **lista_nodi** e le fornisco all'utente.

Funzioni per trovare il percorso critico:

```
587● static int CpFinder(Nodo n) {
588     /*
589     * Funzione scritta per essere lanciata sulle foglie. Restituisce il valore del
590     * critical path che ci servirà per lo scheduling. Va a chiamare in modo
591     * ricorsivo CpRicorsivo.
592     */
593     if (n == null) {
594         return 0;
595     }
596     if (n.isRadice() == true) {
597         return 0;
598     }
599     int v[] = new int[n.getPartenze().size()];
600     for (int i = 0; i < v.length; i++) {
601         v[i] = CpRicorsivo(n.getPartenze().get(i));
602     }
603     return massimo(v);
604 }
605
606● static int CpRicorsivo(Nodo n) {
607     /*
608     * Funzione ricorsiva che esplora i nodi fino alla radice. Serve per calcolare
609     * il critical path
610     */
611     if (n == null) {
612         return 0;
613     }
614     if (n.isRadice() == true) {
615         return 1;
616     }
617     int v[] = new int[n.getPartenze().size()];
618     for (int i = 0; i < v.length; i++) {
619         v[i] = CpRicorsivo(n.getPartenze().get(i));
620     }
621     return (massimo(v) + 1);
622 }
```

Ecco un esempio di output che ho creato usando il file input.txt mostrato in precedenza:

Ho lanciato il programma fornendo 1 componente per ogni tipo di operazione:

```
output - Blocco note di Windows
File Modifica Formato Visualizza ?
Per svolgere tutte le 18 operazioni impiego 11 cicli di clock

Scheduling finale:
Nel ciclo di clock numero 1 verranno effettuate le seguenti operazioni: o a
Nel ciclo di clock numero 2 verranno effettuate le seguenti operazioni: n
Nel ciclo di clock numero 3 verranno effettuate le seguenti operazioni: k b
Nel ciclo di clock numero 4 verranno effettuate le seguenti operazioni: q m e
Nel ciclo di clock numero 5 verranno effettuate le seguenti operazioni: l c
Nel ciclo di clock numero 6 verranno effettuate le seguenti operazioni: d
Nel ciclo di clock numero 7 verranno effettuate le seguenti operazioni: f
Nel ciclo di clock numero 8 verranno effettuate le seguenti operazioni: r h
Nel ciclo di clock numero 9 verranno effettuate le seguenti operazioni: j g
Nel ciclo di clock numero 10 verranno effettuate le seguenti operazioni: i
Nel ciclo di clock numero 11 verranno effettuate le seguenti operazioni: p
```

Ho lanciato il programma fornendo 4 componenti per ogni tipo di operazione:

```
output - Blocco note di Windows
File Modifica Formato Visualizza ?
Per svolgere tutte le 18 operazioni impiego 7 cicli di clock

Scheduling finale:
Nel ciclo di clock numero 1 verranno effettuate le seguenti operazioni: m o c b a
Nel ciclo di clock numero 2 verranno effettuate le seguenti operazioni: l n e
Nel ciclo di clock numero 3 verranno effettuate le seguenti operazioni: k d f
Nel ciclo di clock numero 4 verranno effettuate le seguenti operazioni: q h g
Nel ciclo di clock numero 5 verranno effettuate le seguenti operazioni: r i
Nel ciclo di clock numero 6 verranno effettuate le seguenti operazioni: j
Nel ciclo di clock numero 7 verranno effettuate le seguenti operazioni: p
```

Esempio di scheduling delle operazioni nel primo ciclo di clock con 1 componente per ogni operazione (preso da log_output.txt):

```

Sono al ciclo di clock numero 1
Ho trovato 5 foglie:
Il nodo foglia m ha critical path = 4
Il nodo foglia o ha critical path = 6
Il nodo foglia c ha critical path = 4
Il nodo foglia b ha critical path = 5
Il nodo foglia a ha critical path = 5
Lista delle foglie ordinata per critical path in modo decrescente:
Il nodo foglia o ha critical path = 6
Il nodo foglia b ha critical path = 5
Il nodo foglia a ha critical path = 5
Il nodo foglia m ha critical path = 4
Il nodo foglia c ha critical path = 4
Provo a schedulare tutte le foglie:
Ho schedulato con successo l'operazione o
L'operazione b sarebbe da schedulare ma non ho abbastanza componenti per le somme e sottrazioni
Ho schedulato con successo l'operazione a
L'operazione m sarebbe da schedulare ma non ho abbastanza componenti per le somme e sottrazioni
L'operazione c sarebbe da schedulare ma non ho abbastanza componenti per le somme e sottrazioni
In questo ciclo di clock ho trovato: 4 somme e sottrazioni; 1 moltiplicazioni; 0 divisioni
Elimino i nodi foglia schedulati:
Rimuovo o perché è appena stato schedulato
Rimuovo a perché è appena stato schedulato
Ecco il clone di lista_nodi alla fine del ciclo di clock numero 1
nome=p operazione== radice=true foglia=false cp=0 schedulato=false clock=0 arrivi:i j
nome=j operazione=* radice=false foglia=false cp=0 schedulato=false clock=0 arrivi:k r partenze:p
nome=r operazione=- radice=false foglia=false cp=0 schedulato=false clock=0 arrivi:q l partenze:j
nome=l operazione=* radice=false foglia=false cp=0 schedulato=false clock=0 arrivi:m partenze:r
nome=q operazione=/ radice=false foglia=false cp=0 schedulato=false clock=0 arrivi:k partenze:r
nome=m operazione=- radice=false foglia=true cp=4 schedulato=false clock=0 partenze:l
nome=k operazione=/ radice=false foglia=false cp=0 schedulato=false clock=0 arrivi:n partenze:j q
nome=n operazione+= radice=false foglia=false cp=0 schedulato=false clock=0 partenze:k d
nome=d operazione+= radice=false foglia=false cp=0 schedulato=false clock=0 arrivi:n partenze:g
nome=c operazione+= radice=false foglia=true cp=4 schedulato=false clock=0 partenze:f
nome=b operazione=- radice=false foglia=true cp=5 schedulato=false clock=0 partenze:e
nome=e operazione=* radice=false foglia=false cp=0 schedulato=false clock=0 arrivi:b partenze:h f
nome=h operazione=/ radice=false foglia=false cp=0 schedulato=false clock=0 arrivi:e f partenze:i
nome=i operazione=- radice=false foglia=false cp=0 schedulato=false clock=0 arrivi:h g partenze:p
nome=f operazione+= radice=false foglia=false cp=0 schedulato=false clock=0 arrivi:e c partenze:h g
nome=g operazione+= radice=false foglia=false cp=0 schedulato=false clock=0 arrivi:f d partenze:i

```

Esempio di scheduling delle operazioni nel primo ciclo di clock con 4 componenti per ogni operazione (preso da log_output.txt):

```

Sto entrando nel ciclo do-while che determina lo scheduling delle operazioni

Sono al ciclo di clock numero 1
Ho trovato 5 foglie:
Il nodo foglia m ha critical path = 4
Il nodo foglia o ha critical path = 6
Il nodo foglia c ha critical path = 4
Il nodo foglia b ha critical path = 5
Il nodo foglia a ha critical path = 5
Lista delle foglie ordinata per critical path in modo decrescente:
Il nodo foglia o ha critical path = 6
Il nodo foglia b ha critical path = 5
Il nodo foglia a ha critical path = 5
Il nodo foglia m ha critical path = 4
Il nodo foglia c ha critical path = 4
Provo a schedulare tutte le foglie:
Ho schedulato con successo l'operazione o
Ho schedulato con successo l'operazione b
Ho schedulato con successo l'operazione a
Ho schedulato con successo l'operazione m
Ho schedulato con successo l'operazione c
In questo ciclo di clock ho trovato: 4 somme e sottrazioni; 1 moltiplicazioni; 0 divisioni
Elimino i nodi foglia schedulati:
Rimuovo o perché è appena stato schedulato
Rimuovo b perché è appena stato schedulato
Rimuovo a perché è appena stato schedulato
Rimuovo m perché è appena stato schedulato
Rimuovo c perché è appena stato schedulato
Ecco il clone di lista_nodi alla fine del ciclo di clock numero 1
nome=p operazione== radice=true foglia=false cp=0 schedulato=false clock=0 arrivi:i j
nome=j operazione=* radice=false foglia=false cp=0 schedulato=false clock=0 arrivi:k r partenze:p
nome=r operazione=- radice=false foglia=false cp=0 schedulato=false clock=0 arrivi:q l partenze:j
nome=l operazione=* radice=false foglia=false cp=0 schedulato=false clock=0 partenze:r
nome=q operazione=/ radice=false foglia=false cp=0 schedulato=false clock=0 arrivi:k partenze:r
nome=k operazione=/ radice=false foglia=false cp=0 schedulato=false clock=0 arrivi:n partenze:j q
nome=n operazione+= radice=false foglia=false cp=0 schedulato=false clock=0 partenze:k d
nome=d operazione+= radice=false foglia=false cp=0 schedulato=false clock=0 arrivi:n partenze:g
nome=e operazione=* radice=false foglia=false cp=0 schedulato=false clock=0 partenze:h f
nome=h operazione=/ radice=false foglia=false cp=0 schedulato=false clock=0 arrivi:e f partenze:i
nome=i operazione=- radice=false foglia=false cp=0 schedulato=false clock=0 arrivi:h g partenze:p
nome=f operazione+= radice=false foglia=false cp=0 schedulato=false clock=0 arrivi:e c partenze:h g
nome=g operazione+= radice=false foglia=false cp=0 schedulato=false clock=0 arrivi:f d partenze:i

```