Національний технічний університет України

«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

Організація обчислювальних процесів

Лабораторна робота №4

**«Дослідження принципів проектування та роботи динамічного та статичного планування»**

Виконала:

студентка групи ІВ-71

Молчанова В.С.

Номер залікової книжки: 7110

Перевірив:

Сімоненко А.В.

Київ

2020 р.

## Завдання на роботу

Написати програму, що реалізує один з алгоритмів статичного чи динамічного планування, згідно з варіантом.

## Варіант

1. 10 mod 13 + 1 = 11 – Скласти програму виділення вершин, які мають ознаку не явної транзитності.

### Опис роботи алгорітму

На первом шаге алгоритма строится базовое решение. Под базовым решением понимается выделение отдельного процессора для каждой задачи. Такое решение используется в качестве основного, от которого производится поиск лучшего решения.

Основными атрибутами, по которым осуществляется планирование для DAG, являются *t-level* и *b-level* [1,6].

t-level – это самый длинный путь от начальной вершины до данной вершины ni , без учета веса ni. Путь считается путем сложения всех весов вершин и пересылок, через которые он проходит. Этот параметр так же отражает наименьшее время начала вершины. T-level для i-й вершины вычисляется по следующей формуле:

*tlevel(ni)= max (tlevel(nm)+ wm + cm,i),*

где nm=pred(ni) – предшествующие вершины, wm – вычислительная сложность, cm,i – временные затраты на коммуникацию.

b-level - это самый длинный путь от вершины ni до исходной. Так же выделяют статический b-level, в котором не учитываются пересылки. B-level для i-го узла вычисляется по формуле:

*blevel(ni)= wi+ max (blevel(nm)+ cm,i),*

где nm=succ(ni) – последующие вершины, wi – вычислительная сложность, cm,i – временные затраты на коммуникацию. А статический b-level вычисляется по формуле:

*sblevel(ni)= wi+ max (sblevel(nm)),*

где nm=succ(ni) – последующие вершины, wi – вычислительная сложность.

 Стоит отметить, что данные необходимо вычислять на каждом шаге алгоритма только для тех вершин, которые находятся на последующем уровне, при нисходящем планировании, и тех, которые находятся уровнем выше при восходящем. Это позволяет исключить повторное полное сканирование графа, что значительно ускоряет процесс планирования.

Обозначим множество вершин, принадлежащих k-му уровню:

*nk={nki}, i=1,mk,*

где nki – i-я вершина k-го уровня, mk – количество вершин на k-ом уровне. Если вершина nki имеет единственную инцидентную ей дугу e, и вершина nk-1i так же имеет единственную дугу e инцидентную ей, то вершины nki и nk-1iнеобходимо кластеризовать.

При нисходящем планировании на каждом шаге алгоритма выполняются следующие действия:

* Вычисляется t-level для каждой вершины nk+1j, находящейся на следующем уровне.
* Рассматривается множество задач nk находящихся на одном (k) уровне. Для каждой задачи k-го уровня выбирается множество задач nk+1j, j=1,mk+1, находящихся на k+1 уровне, имеющих дугу, инцидентную рассматриваемой вершине nkj, т.е. вершина nk+1j имеет информационную зависимость от вершины nki.
* Для кластеризации с nki выбирается та вершина nk+1j, которая имеет наибольший t-level:max(t-level(nk+1j)). Остальные вершины, имеющие информационную зависимость от рассматриваемой остаются без изменений, однако в последствии могут быть кластеризованы с другими вершинами k-го уровня.

При восходящем планировании на каждом шаге алгоритма выполняются следующие действия:

* Вычисляется b-level для каждой вершины nk-1j, находящейся на предыдущем уровне.
* Рассматривается множество задач nk находящихся на одном (k) уровне. Для каждой задачи k-го уровня выбирается множество задач nk-1j, j=1,mk+1, находящихся на k-1 уровне, имеющих дугу, инцидентную рассматриваемой вершине nki, т.е. вершина nki имеет информационную зависимость от вершины nk-1j.
* Для кластеризации с nki выбирается та вершина nk-1j, которая имеет наибольший b-level: max(b-level(nk-1j)).

Для кластеризованных вершин, находящихся в одном узле, время затрачиваемое на коммуникацию приравниваем к 0.

Так же при планировании стоит учитывать вершины, обладающие свойствами неявной транзитности, т. е. для nki вершины выполняется неравенство:

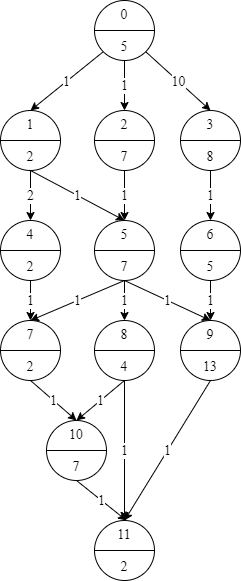
*tlevel(nki) ≥ min(tlevel(nk+1j)).*

Такие вершины при планировании могут быть кластеризованы с вершинами, находящимися на более низком уровне, чем k+1. Зачастую это может привести к уменьшению ресурсов, требуемых для текущей задачи. Такую вершину целесообразно перенести на другой уровень для кластеризации, если выполняется соотношение:

*ntlevel(nki) ≤ max(tlevel(nkj)),*

где ntlevel – tlevel для нового уровня.

Розглянемо приклад



Порахуємо *tlevel* для кожної вершини:

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
| 0 | 6 | 6 | 15 | 10 | 14 | 24 | 22 | 22 | 30 | 27 | 44 |

Виходячи з формули визначення вершин з ознакою неявної транзитності

*tlevel(nki) ≥ min(tlevel(nk+1j))*

отримаємо такі вершини:

3, 6, 9

## Результати роботи програми

Матриця зв'язності:

0 1 1 10 0 0 0 0 0 0 0 0

0 0 0 0 2 1 0 0 0 0 0 0

0 0 0 0 0 1 0 0 0 0 0 0

0 0 0 0 0 0 1 0 0 0 0 0

0 0 0 0 0 0 0 1 0 0 0 0

0 0 0 0 0 0 0 1 1 1 0 0

0 0 0 0 0 0 0 0 0 1 0 0

0 0 0 0 0 0 0 0 0 0 1 0

0 0 0 0 0 0 0 0 0 0 1 0

0 0 0 0 0 0 0 0 0 0 0 1

0 0 0 0 0 0 0 0 0 0 0 1

0 0 0 0 0 0 0 0 0 0 0 0

=============================================

Ваги вершин:

5 2 7 8 2 7 5 2 4 13 7 2

=============================================

Розбиття по рівням та розрахунок t\_level (формат запису '<номер вершини>': <t\_level>)

'0': 0

'1': 6 | '2': 6 | '3': 15

'4': 10 | '5': 14 | '6': 24

'7': 22 | '8': 22 | '9': 30

'10': 27

'11': 44

=============================================

Вершини, що мають ознаку неявної транзитності:

'3': 15 | '6': 24 | '9': 30

Можна побачити, що результат, виведений програмою збігається з розрахованим раніше.

## Лістинг програми

**class** Node:  
 **def** \_\_init\_\_(self, idx, weight):  
 self.idx = idx  
 self.weight = weight  
 self.t\_level = 0  
  
 **def** \_\_str\_\_(self):  
 **return "'{0}': {2}"**.format(self.idx, self.weight, self.t\_level)  
  
 **def** \_\_repr\_\_(self):  
 **return "'{0}': {2}"**.format(self.idx, self.weight, self.t\_level)  
  
  
data = []  
**with** open(**"data.txt"**) **as** f:  
 **for** line **in** f:  
 data.append([int(x) **for** x **in** line.split()])  
  
matrix = data[:-1]  
weights = data[-1]  
n = len(matrix)  
nodes = [Node(i, weights[i]) **for** i **in** range(n)]  
  
  
**def** get\_levels():  
 levels = [[nodes[0]]]  
 current\_level = 1  
 free\_nodes = list(nodes[1:])  
 **while** len(free\_nodes) > 0:  
 levels.append([])  
 **for** free\_node **in** free\_nodes:  
 to\_current\_level = **False** *# if current free node doesn't connected to any node in previous level, skip it  
 # also calculating t\_level* **for** prev\_node **in** levels[current\_level - 1]:  
 edge\_weight = matrix[prev\_node.idx][free\_node.idx]  
 **if** edge\_weight > 0:  
 to\_current\_level = **True** free\_node.t\_level = max(free\_node.t\_level, prev\_node.t\_level + prev\_node.weight + edge\_weight)  
 **if not** current\_level:  
 **continue** *# if current free node is connected to any at current level, skip it* **for** node **in** levels[current\_level]:  
 **if** matrix[free\_node.idx][node.idx] > 0 **or** matrix[node.idx][free\_node.idx] > 0:  
 to\_current\_level = **False  
 break  
  
 if** to\_current\_level:  
 levels[current\_level].append(free\_node)  
  
 free\_nodes = free\_nodes[len(levels[current\_level]):]  
 current\_level += 1  
  
 **return** levels  
  
  
**def** get\_transite(levels: list):  
 transite = []  
 minimums = list(map(**lambda** level: min([node.t\_level **for** node **in** level]), levels))  
 **for** i **in** range(len(levels) - 1):  
 **for** current\_node **in** levels[i]:  
 **if** current\_node.t\_level >= minimums[i + 1]:  
 transite.append(current\_node)  
 **return** transite  
  
  
**def** main():  
 print(**"Матриця зв'язності: "**)  
 **for** row **in** matrix:  
 **for** e **in** row:  
 print(**"{0:<4}"**.format(e), end=**''**)  
 print()  
 print(**"="** \* (4 \* n - 3))  
 print(**"Ваги вершин: "**)  
 **for** w **in** weights:  
 print(**"{0:<4}"**.format(w), end=**''**)  
 print()  
 print(**"="** \* (4 \* n - 3))  
  
 print(**"Розбиття по рівням та розрахунок t\_level (формат запису '<номер вершини>': <t\_level>)"**)  
 levels = get\_levels()  
 **for** level **in** levels:  
 print(**" | "**.join(map(**lambda** node: **"{0:<8}"**.format(str(node)), level)))  
 print(**"="** \* (4 \* n - 3))  
  
 print(**"Вершини, що мають ознаку неявної транзитності:"**)  
 transite = get\_transite(levels)  
 print(**" | "**.join(map(**lambda** node: **"{0:<8}"**.format(str(node)), transite)))  
  
  
main()