MAXimal

added: 10 Jun 2008 19:27
Edited: 9 Jul 2009 18:24

Search Strongly Connected Components, construction condensation graph

Contents [hide]

Definitions, statement of the problem

Given a directed graph Gwhose vertex set $V$and the set of edges - E. Loops and multiple edges are allowed. We denote $n$the number of vertices, through m- the number of edges.

Strongly connected component (strongly connected component) is called a (maximum respect to inclusion) subset of vertices Csuch that any two vertices of this subset reachable from each other, ie, for $\forall u, v \in C$:

$$u \mapsto v, v \mapsto u$$

where the symbol $\mapsto$hereafter we shall denote the attainability, ie existence of a path from the first vertex to the second.

It is clear that strong connectivity components for this graph do not intersect, ie in fact it is a partition of all vertices of the graph. Hence the logical definition of **condensation** $G^{\rm SCC}$ as the graph obtained from the compression of the graph of each component of the strong connectivity in a single vertex. Each vertex of the graph corresponds to the condensation component strongly connected graph G, and a directed edge between two vertices $C_i$and $C_j$graph condensation is carried out, if there exists a pair of vertices $u \in C_i, v \in C_j$between which there was an edge in the original graph, ie $(u, v) \in E$.

The most important property of the graph condensation is that it **is acyclic** . Indeed, suppose that $C \mapsto C'$we prove that $C' \not\mapsto C$. From the definition of condensation we find that there are two vertices $u \in C$and $v \in C'$that $u \mapsto v$. Will prove by contradiction, ie, assume that $C' \mapsto C$, then there are two vertices $u' \in C$and $v' \in C'$that $v' \mapsto u'$. But since $u$and $u'$are in the same strongly connected component, then there is a path between them; similarly for $v$and v'. As a result, combining the way, we see that $v \mapsto u$, at the same time $u \mapsto v$. Consequently, u and $v$must belong to the same strongly connected component, ie, a contradiction, as required.

Algorithm described below are shown in this graph all strongly connected components. Build on them graph condensation is not difficult.

Algorithm

Algorithm described here has been proposed independently Kosarayu (Kosaraju) and Sharir (Sharir) in 1979, It is very easy to implement algorithm based on two series of depth-first search , and therefore the running time $O(n+m)$.

In the first step of the algorithm, a series of detours in depth, visiting the whole graph. To do this, we go through all the vertices of the graph and from each vertex is not being visited call bypassing in depth. In this case, for each vertex $v$remember **time to** $\mathrm{tout}[v]$. These retention times play a key role in the algorithm, and this role is expressed in the below theorem.

First, we introduce the notation: time- $\mathrm{tout}[C]$components of $C$the strong connectivity is defined as the maximum of the values $\mathrm{tout}[v]$for all $v \in C$. In addition, the proof of the theorem will be referred to the time and input to each vertex $\mathrm{tin}[v]$, and similarly define the time of entry $\mathrm{tin}[C]$for each strongly connected component of a minimum of the values $\mathrm{tin}[v]$for all $v \in C$.

Theorem . Let $C$and C'- two different strongly connected components, and let the graph condensation between an edge (C, C'). Then $\mathrm{tout}[C] > \mathrm{tout}[C']$.

In the proof there are two fundamentally different cases depending on which of the components of the first tour will go in depth, ie, Depending on the ratio between $\mathrm{tin}[C]$and $\mathrm{tin}[C']$:

- The first component was reached C. This means that at some point in time in depth tour comes to a vertex $v$components C, and all the other vertices of the components $C$and C'not yet visited. However, since by the condition in the graph there is an edge of the condensations (C, C'), then from the top $v$will be achieved not only the entire component C, but also the entire component C'. This means that when you start from the top $v$bypassing deep pass over all vertices of the components $C$and C', and, therefore, they will be in relation to the descendants $v$in the tree traversal in depth, ie, for each vertex $u \in C \cup C', u \neq v$is satisfied $\mathrm{tout}[v] > \mathrm{tout}[u]$, qed
- The first component was reached C'. Again, at some point of time in depth tour comes to a vertex $v \in C'$, with all the other vertices of the component $C$and C' are visited. By assumption, the graph condensations existed edge (C, C'), is due to condensation acyclic graph, there is no way back $C' \not\rightarrowtail C$, ie bypass in depth from the top $v$you reach the top C. This means that they will be visited by a circuit in depth later, whence $\mathrm{tout}[C] > \mathrm{tout}[C']$, qed

The above theorem is the **basis for the algorithm** search Strongly Connected Components. From it follows that any edge (C, C')in the graph goes condensations of components with greater magnitude toutin the component with a lower value.

If we sort all the vertices $v \in V$in descending order of exit time $\mathrm{tout}[v]$, the first will be a vertex $u$belonging to the "root" strongly connected component, ie, which is not part of one edge of the graph condensations. Now we would like to start a tour of this peak u, which would be attended only this component is strongly connected and not gone to any other; learning how to do it, we can gradually select all strongly connected components: removing from the graph vertex of the first selected component, again, we find among the remaining node with the highest value tout, re-run of it this tour, etc.

To learn how to do this tour, consider **the transpose of the graph** G^T , ie the graph obtained from G each change in direction of the opposite edge. It is not difficult to understand that in this graph are the same strongly connected components as in the original graph. Moreover, the graph condensation $(G^T)^{\text{SCC}}$ for him is equal transposed graph condensation of the original graph G^{SCC}. This means that from now we are considering the "root" component will not be leaving the edges to other components.

So, to get around the whole "root" strongly connected component containing a vertex v , enough to run the tour from the vertex v in the graph G^T. This tour will visit all the vertices of the strongly connected components, and only them. As already mentioned, then we can mentally remove these vertices of the graph, find the next vertex with the maximum value $\text{tout}[v]$ and run circumvention transposed graph of it, etc.

Thus we have constructed the following **algorithm** selection Strongly Connected Components:

Step 1. Launch a series of detours in depth graph G, which returns the top in order of increasing time-to tout, ie some list order.

Step 2. Build transposed graph G^T. Launch a series of detours in depth / width of the graph in the order defined by the list order (ie, in the reverse order, ie, in order to reduce the time of output). Each set of vertices reached as a result of the next start crawling, and there will be another component of strong connectivity.

Asymptotic behavior of the algorithm, obviously, is $O(n+m)$ because it is only two round trips in depth / width.

Finally, it is appropriate to note the relationship with the notion of **topological sorting** . First, step 1 of the algorithm is not nothing but a topological sort of the graph G (in fact, that is exactly what is sorting the vertices retention time). Secondly, the scheme of the algorithm is that both components are strongly connected it generates in the order of decreasing since the release, so it generates components - the vertices of condensation in topological sort order.

Implementation

```
vector < vector<int> > g, gr;
vector<char> used;
vector<int> order, component;

void dfs1 (int v) {
        used[v] = true;
        for (size_t i=0; i<g[v].size(); ++i)
                if (!used[ g[v][i] ])
                        dfs1 (g[v][i]);
        order.push_back (v);
}

void dfs2 (int v) {
        used[v] = true;
        component.push_back (v);
```

```
		for (size_t i=0; i<gr[v].size(); ++i)
			if (!used[ gr[v][i] ])
				dfs2 (gr[v][i]);
}

int main() {
		int n;
		... чтение n ...
		for (;;) {
			int a, b;
			... чтение очередного ребра (a,b) ...
			g[a].push_back (b);
			gr[b].push_back (a);
		}

		used.assign (n, false);
		for (int i=0; i<n; ++i)
			if (!used[i])
				dfs1 (i);
		used.assign (n, false);
		for (int i=0; i<n; ++i) {
			int v = order[n-1-i];
			if (!used[v]) {
				dfs2 (v);
				... вывод очередной component ...
				component.clear();
			}
		}
}
```

Here $\rm g$kept the Count himself, and $\rm gr$- transposed graph. Function $\rm dfs1$crawls deep in the graph G, the function $\rm dfs2$- for transposed G^T. Function $\rm dfs1$populates the list $\rm order$vertices in order of increasing time out (in fact, doing a topological sort). Function $\rm dfs2$saves all reached the top of the list $\rm component$, which after each run will contain another component of strong connectivity.

Literature

- Thomas Cormen, Charles Leiserson, Ronald Rivest, Clifford Stein. **Algorithms: Design and Analysis** [2005]
- M. Sharir. **A strong-Connectivity algorithm and ITS applications in Data-Flow analysis** [1979]

10 Комментариев e-maxx Войти

Лучшее вначале Поделиться Избранный

Присоединиться к обсуждению...

Артём Финк • месяц назад

Я кое что не понимаю
Компонента сильной связанности может быть циклом
А топологическую сортировку для графа с циклом выполнить нельзя
Следовательно dfs1 является просто копией топологической сортировки
Но по сути так не называется так как противоречит определению топологической сортировки

• Ответить • Поделиться ›

dk • 4 месяца назад

Не будет ли лучше вместо vector<int> order использовать стек, а затем идти в прямом порядке?

• Ответить • Поделиться ›

Oren Leto → dk • 3 месяца назад

Для ускорения скорее можно завести простой массив. Стек не даст никакого прироста в производительности

1 • Ответить • Поделиться ›

ternsip • год назад

Вы использовали vector<char> used; А почему бы не vector<bool> used; ?

• Ответить • Поделиться ›

Ляо Си → ternsip • 10 месяцев назад

vector<bool> работает медленнее из-за битового сжатия

2 • Ответить • Поделиться ›

Axiom • 2 года назад

Действительно, не ясно: ведь если компонента связности является листом, то из ее вершин нельзя попасть в другие компоненты, а значит и в транспонированном графе нет необходимости.
То есть имеет смысл рассматривать вершину не с наибольшим tout, как в алгоритме, а с наименьшим. И от нее запускать алгоритм обхода графа G.

• Ответить • Поделиться ›

e_maxx Модератор → Axiom • 2 года назад

Я не понимаю, о чём вы говорите. Рассмотрим граф: три вершины, три ребра: 1->2, 2->1, 1->3. Описанный в статье алгоритм даст order=[2,3,1], запустит dfs2(1) и dfs2(3), найдя правильное решение: компоненты [1,2] и [3]. Если же, как вы говорите, запускать из вершины с наименьшим tout (т.е. 2), то обход по графу G посетит все три вершины и найдёт неправильный ответ: одну компоненту [1,2,3].

1 • Ответить • Поделиться ›

Axiom ➜ e_maxx • 2 года назад

Спасибо!
Теперь все ясно: наибольший tout гарантирует корневую компоненту графа конденсации, тогда как наименьший tout вообще ничего не означает.

• Ответить • Поделиться ›

RS • 2 года назад

Честно говоря, не пойму зачем нужно усложнять объяснение введением транспонированного графа. Сразу ясно, что надо брать вершину v с