добавлено: 6 Jul 2008 9:33
редактировано: 7 Sep 2009 9:59

Задача 2-SAT

Содержание [скрыть]

Задача 2-SAT (2-satisfiability) - это задача распределения значений булевым переменным таким образом, чтобы они удовлетворяли всем наложенным ограничениям.

Задачу 2-SAT можно представить в виде конъюнктивной нормальной формы, где в каждом выражении в скобках стоит ровно по две переменной; такая форма называется 2-CNF (2-conjunctive normal form). Например:

```
(a || c) && (a || !d) && (b || !d) && (b || !e) && (c || d)
```

Приложения

Алгоритм для решения 2-SAT может быть применим во всех задачах, где есть набор величин, каждая из которых может принимать 2 возможных значения, и есть связи между этими величинами:

- **Расположение текстовых меток на карте или диаграмме**.
  Имеется в виду нахождение такого расположения меток, при котором никакие две не пересекаются.
  Стоит заметить, что в общем случае, когда каждая метка может занимать множество различных позиций, мы получаем задачу general satisfiability, которая является NP-полной. Однако, если ограничиться только двумя возможными позициями, то полученная задача будет задачей 2-SAT.
- **Расположение рёбер при рисовании графа**.
  Аналогично предыдущему пункту, если ограничиться только двумя возможными способами провести ребро, то мы придём к 2-SAT.
- **Составление расписания игр**.
  Имеется в виду такая система, когда каждая команда должна сыграть с каждой по одному разу, а требуется распределить игры по типу домашняя-выездная, с некоторыми наложенными ограничениями.
- и т.д.

Алгоритм

Сначала приведём задачу к другой форме - так называемой импликативной форме. Заметим, что выражение вида a || b эквивалентно !a => b или !b => a. Это можно воспринимать следующим образом: если есть выражение a || b, и нам необходимо добиться обращения его в true, то, если a=false, то необходимо b=true, и наоборот, если b=false, то необходимо a=true.

Построим теперь так называемый **граф импликаций**: для каждой переменной в графе будет по две вершины, обозначим их через x_i и $!x_i$. Рёбра в графе будут соответствовать импликативным связям.

Например, для 2-CNF формы:

```
(a || b) && (b || !c)
```

Граф импликаций будет содержать следующие рёбра (ориентированные):

```
!a => b
!b => a
!b => !c
c => b
```

Стоит обратить внимание на такое свойство графа импликаций, что если есть ребро a => b, то есть и ребро !b => !a.

Теперь заметим, что если для какой-то переменной x выполняется, что из x достижимо !x, а из !x достижимо x, то задача решения не имеет. Действительно, какое бы значение для переменной x мы бы ни выбрали, мы всегда придём к противоречию - что должно быть выбрано и обратное ему значение. Оказывается, что это условие является не только достаточным, но и необходимым (доказательством этого факта будет описанный ниже алгоритм). Переформулируем данный критерий в терминах теории графов. Напомним, что если из одной вершины достижима другая, а из той вершины достижима первая, то эти две вершины находятся в одной сильно связной компоненте. Тогда мы можем сформулировать **критерий существования решения** следующим образом:

Для того, чтобы данная задача 2-SAT **имела решение**, необходимо и достаточно, чтобы для любой переменной x вершины x и !x находились **в разных компонентах сильной связности** графа импликаций.

Этот критерий можно проверить за время O (N + M) с помощью алгоритма поиска сильно связных компонент.

Теперь построим собственно **алгоритм** нахождения решения задачи 2-SAT в предположении, что решение существует.

Заметим, что, несмотря на то, что решение существует, для некоторых переменных может выполняться, что из x достижимо !x, или (но не одновременно), из !x достижимо x. В таком случае выбор одного из значений переменной x будет приводить к противоречию, в то время как выбор другого - не будет. Научимся выбирать из двух значений то, которое не приводит к возникновению противоречий. Сразу заметим, что, выбрав какое-либо значение, мы должны запустить из него обход в глубину/ширину и пометить все значения, которые следуют из него, т.е. достижимы в графе импликаций. Соответственно, для уже помеченных вершин никакого выбора между x и !x делать не нужно, для них значение уже выбрано и зафиксировано. Нижеописанное правило применяется только к непомеченным ещё вершинам.

Утверждается следующее. Пусть comp[v] обозначает номер компоненты сильной связности, которой принадлежит вершина v, причём номера упорядочены в порядке топологической сортировки компонент сильной связности в графе компонентов (т.е. более ранним в порядке топологической сортировки соответствуют большие номера: если есть путь из v в w, то comp[v] <= comp[w]). Тогда, если comp[x] < comp[!x], то выбираем значение !x, иначе, т.е. если comp[x] > comp[!x], то выбираем x.

Докажем, что при таком выборе значений мы не придём к противоречию. Пусть, для определённости, выбрана вершина x (случай, когда выбрана вершина !x, доказывается симметрично).

Во-первых, докажем, что из x не достижимо !x. Действительно, так как номер компоненты сильной связности comp[x] больше номера компоненты comp[!x], то это означает, что компонента связности, содержащая x, расположена левее компоненты связности, содержащей !x, и из первой никак не может быть достижима последняя.

Во-вторых, докажем, что никакая вершина y, достижимая из x, не является "плохой", т.е. неверно, что из y достижимо !y. Докажем это от противного. Пусть из x достижимо y, а из y достижимо !y. Так как из x достижимо y, то, по свойству графа импликаций, из !y будет достижимо !x. Но, по предположению, из y достижимо !y. Тогда мы получаем, что из x достижимо !x, что противоречит условию, что и требовалось доказать.

Итак, мы построили алгоритм, который находит искомые значения переменных в предположении, что для любой переменной x вершины x и !x находятся в разных компонентах сильной связности. Выше показали корректность этого алгоритма. Следовательно, мы одновременно доказали указанный выше критерий существования решения.

Теперь мы можем собрать **весь алгоритм** воедино:

- Построим граф импликаций.
- Найдём в этом графе компоненты сильной связности за время O (N + M), пусть comp[v] - это номер компоненты сильной связности, которой принадлежит вершина v.
- Проверим, что для каждой переменной x вершины x и !x лежат в разных компонентах, т.е. comp[x] ≠ comp[!x]. Если это условие не выполняется, то вернуть "решение не существует".
- Если comp[x] > comp[!x], то переменной x выбираем значение true, иначе - false.

Реализация

Ниже приведена реализация решения задачи 2-SAT для уже построенного графа импликаций g и обратного ему графа gt (т.е. в котором направление каждого ребра изменено на противоположное).

Программа выводит номера выбранных вершин, либо фразу "NO SOLUTION", если решения не существует.

```
int n;
vector < vector<int> > g, gt;
vector<bool> used;
vector<int> order, comp;

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

void dfs2 (int v, int cl) {
	comp[v] = cl;
	for (size_t i=0; i<gt[v].size(); ++i) {
		int to = gt[v][i];
		if (comp[to] == -1)
			dfs2 (to, cl);
```

```
	}
}

int main() {
	... чтение n, графа g, построение графа gt ...

	used.assign (n, false);
	for (int i=0; i<n; ++i)
		if (!used[i])
			dfs1 (i);

	comp.assign (n, -1);
	for (int i=0, j=0; i<n; ++i) {
		int v = order[n-i-1];
		if (comp[v] == -1)
			dfs2 (v, j++);
	}

	for (int i=0; i<n; ++i)
		if (comp[i] == comp[i^1]) {
			puts ("NO SOLUTION");
			return 0;
		}
	for (int i=0; i<n; ++i) {
		int ans = comp[i] > comp[i^1] ? i : i^1;
		printf ("%d ", ans);
	}

}
```