

Problema do fluxo máximo

Método do pré-fluxo

Fila de vértices ativos

Juliana Barby Simão
APOIO FINANCEIRO DA FAPESP
PROCESSO 04/00580-8

Marcelo Hashimoto
APOIO FINANCEIRO DA FAPESP
PROCESSO 04/00581-4

ORIENTADOR: José Coelho de Pina

Sumário

1. Introdução	2
2. Descrição	2
3. Compilação e execução	2
4. Referências	2
5. Método do pré-fluxo: algoritmo da fila de vértices ativos	3
15. Execução de um relabel	7
16. Execução de um push	7
18. Fila de vértices ativos	9
19. Função principal	11
20. Consistência dos parâmetros	11
25. Impressão do fluxo de intensidade máxima	13
26. Impressão do separador de capacidade mínima	13
28. Estrutura geral	15
29. Bibliotecas	15
30. Macros	15

1. Introdução

Esta é uma implementação em CWEB-L^AT_EX do **algoritmo da fila de vértices ativos**, uma versão do **método do pré-fluxo** para resolver o **problema do fluxo máximo**. A plataforma SGB é necessária para a execução.

2. Descrição

Este programa recebe o nome de um arquivo que contém um grafo no formato SGB, o nome de um arquivo de saída, o nome de um vértice fonte e o nome de um vértice sorvedouro e imprime no arquivo de saída um fluxo de intensidade máxima e um separador de capacidade mínima da rede representada pelo grafo. Assume-se que as capacidades dos arcos estão representadas no campo *len*. A opção `-d` também pode ser informada, indicando que os rótulos dos vértices devem ser inicializados com a distância real entre cada vértice e o sorvedouro.

3. Compilação e execução

```
make fildeativos.tex para gerar o arquivo LATEX de documentação.  
make fildeativos.dvi para gerar o arquivo DVI de visualização.  
make fildeativos.pdf para gerar o arquivo PDF de visualização.  
make fildeativos.ps para gerar o arquivo PostScript de visualização.  
make fildeativos.c para gerar o código-fonte C do programa.  
make fildeativos para gerar o executável do programa.  
fildeativos para executar o programa.
```

4. Referências

Sobre a plataforma SGB:

<http://www-cs-faculty.stanford.edu/~knuth/sgb.html>

Sobre a linguagem de *literate programming* CWEB-L^AT_EX:

<http://www-cs-faculty.stanford.edu/~knuth/cweb.html>

Sítio do projeto:

<http://www.ime.usp.br/~coelho/oticomb/>

5. Método do pré-fluxo: algoritmo da fila de vértices ativos

O método do pré-fluxo recebe um grafo, representando uma rede capacitada, dois vértices s e t e devolve um st -fluxo de intensidade máxima e um st -separador de capacidade mínima nessa rede. Os vértices s e t são, respectivamente, a *fonte* e o *sorvedouro*. O st -separador mínimo é um certificado para a maximalidade do fluxo encontrado. O método começa a partir de um pré-fluxo inicial e em cada iteração escolhe um vértice ativo para aplicar uma operação push ou uma operação relabel. Quando não existem mais vértices ativos, o método pára.

Devido ao interesse na complexidade experimental do algoritmo, imprime-se o número total de iterações após a execução. Como a rede originalmente não contém arcos irmãos, eles devem ser construídos antes do início da primeira iteração, para a obtenção da rede residual.

O algoritmo da fila de vértices ativos ordena os vértices ativos através de uma fila FIFO (*first-in-first-out*) e efetua pushes saturadores sobre arcos admissíveis incidentes ao primeiro vértice da fila, até que o mesmo deixe de ser ativo ou sofra um relabel.

```
<Algoritmo da fila de vértices ativos 5> ≡
void fila_de_ativos(Graph *g, Vertex *fonte, Vertex *sorvedouro,
                    boolean distancia)
{
  <Variáveis da função fila_de_ativos 17>
  <Inicializa fluxo 6>
  <Constrói arcos irmãos 7>
  <Executa pré-processamento 9>
  iteracoes = 0;
  <Insere vértices iniciais na fila 12>
  while (!fila_vazia()) {
    i = primeiro_fila();
    <Executa push ou relabel 13>
    iteracoes++;
  }
  fprintf(stdout, "Número de iterações: %d\n", iteracoes);
  return;
}
```

Este código é usado no bloco 28.

6. Antes de construir os arcos irmãos, o fluxo em cada arco da rede original é inicializado com zero. Aproveitamos também para inicializar o excesso em cada vértice, bem como o apontador para o arco corrente em sua lista de adjacências.

```
<Inicializa fluxo 6> ≡
for (i = g-vertices; i < g-vertices + g-n; i++) {
  for (a = i-arcs; a; a = a-next) {
    a-flx = 0;
  }
}
```

```

     $i \rightarrow exc = 0;$ 
     $i \rightarrow atual = i \rightarrow arcs;$ 
}

```

Este código é usado no bloco 5.

7. Os arcos irmãos são construídos exatamente segundo sua definição. Note que o arco irmão gerado é inicialmente apontado por $j \rightarrow arcs$.

```

< Constrói arcos irmãos 7 >  $\equiv$ 
  for ( $i = g \rightarrow vertices; i < g \rightarrow vertices + g \rightarrow n; i++$ ) {
    for ( $a = i \rightarrow arcs; a; a = a \rightarrow next$ ) {
      if ( $arco\_original(a)$ ) {
         $j = a \rightarrow tip;$ 
         $gb\_new\_arc(j, i, a \rightarrow cap);$ 
         $a \rightarrow irmao = j \rightarrow arcs;$ 
         $a \rightarrow irmao \rightarrow flux = -1;$ 
         $a \rightarrow irmao \rightarrow irmao = a;$ 
      }
    }
  }
}

```

Este código é usado no bloco 5.

8. Nesta implementação, os arcos irmãos dos arcos da rede original são reconhecidos por terem fluxo negativo.

```

< Função arco\_original 8 >  $\equiv$ 
  int arco\_original( $Arc * a$ )
  {
    return ( $a \rightarrow flux \geq 0$ );
  }

```

Este código é usado no bloco 28.

9. O pré-processamento consiste em atribuir o pré-fluxo inicial e os rótulos-distância iniciais.

```

< Executa pré-processamento 9 >  $\equiv$ 
  < Define pré-fluxo inicial 10 >
  < Define rótulos distância 11 >

```

Este código é usado no bloco 5.

10. O pré-fluxo inicial x é tal que $x_a = u_a$ para todo arco a que sai do vértice fonte e $x_a = 0$ para todo arco a restante do grafo. Como o fluxo em cada arco já foi inicializado com zero, então resta definir o fluxo nos arcos que saem do vértice fonte.

```

⟨ Define pré-fluxo inicial 10 ⟩ ≡
  for (a = fonte→arcs; a; a = a→next) {
    if (arco_original(a)) {
      if (a→tip ≠ fonte) {
        a→flx = a→cap;
        fonte→exc -= a→flx;
        a→tip→exc += a→flx;
      }
    }
  }
}

```

Este código é usado no bloco 9.

11. Por padrão, a função-distância inicial d é tal que $d(fonte) = n$ e $d(i) = 0$, para todo vértice i restante. Entretanto, se a opção *distancia* estiver ativa, então $d(i)$ é a distância entre i e o vértice sorvedouro na rede, para todo vértice i diferente da fonte. Nesse caso, uma busca em largura reversa, isto é, a partir do vértice sorvedouro e utilizando apenas arcos irmãos, é efetuada na rede para a definição dos rótulos.

```

⟨ Define rótulos distância 11 ⟩ ≡
  for (i = g→vertices; i < g→vertices + g→n; i++) {
    if (¬distancia) i→dist = 0;
    else i→dist = -1;
  }
  fonte→dist = g→n;
  if (distancia) {
    inicializa_fila(g);
    insere_fila(sorvedouro);
    sorvedouro→dist = 0;
    while (¬fila_vazia()) {
      i = remove_fila();
      for (a = i→arcs; a; a = a→next) {
        if (¬arco_original(a)) {
          j = a→tip;
          if (j→dist ≡ -1) {
            j→dist = i→dist + 1;
            insere_fila(j);
          }
        }
      }
    }
  }
}

```

Este código é usado no bloco 9.

12. Os vértices ativos obtidos no pré-processamento são adicionados à fila. Deve-se adicionar uma condição extra para não adicionar o sorvedouro.

```

< Insere vértices iniciais na fila 12 > ≡
  inicializa_fila(g);
  for (i = g-vertices; i < g-vertices + g-n; i++) {
    if (i ≠ sorvedouro ∧ i-exc > 0) insere_fila(i);
  }

```

Este código é usado no bloco 5.

13. Se o vértice ativo i examinado é origem de algum arco admissível a , executa-se um push no arco a . Senão, executa-se um relabel no vértice i . Deve-se examinar apenas arcos da rede residual, isto é, com capacidade residual positiva.

```

< Executa push ou relabel 13 > ≡
  for (a = i-atual; a; a = a-next) {
    if (get_cap_residual(a) > 0) {
      if (i-dist ≡ a-tip-dist + 1) break;
    }
  }
  if (a ≡ Λ) {
    < Executa um relabel 15 >
    i-atual = i-arcs;
  }
  else {
    < Executa um push 16 >
    i-atual = a;
  }

```

Este código é usado no bloco 5.

14. A capacidade residual de um arco da rede original corresponde à diferença entre sua capacidade e seu fluxo corrente. A capacidade residual de um arco irmão de um arco original corresponde ao fluxo corrente em seu arco irmão.

```

< Função get_capacidade_residual 14 > ≡
  long get_cap_residual(Arc * a)
  {
    if (¬ arco_original(a)) {
      return a-irmao-flx;
    }
    else {
      return (a-cap - a-flx);
    }
  }

```

Este código é usado no bloco 28.

15. Execução de um relabel

Para executar um relabel é necessário visitar todos os vizinhos do vértice examinado i na rede residual. Ao invés de manter uma estrutura de dados separada para a rede residual, mantemos esta implícita. Para tanto, basta que a busca considere apenas os arcos com capacidade residual positiva. O vértice que sofre um relabel deve ser sempre movido para o final da fila de vértices ativos.

```
⟨ Executa um relabel 15 ⟩ ≡
  for ( $min = -1, a = i \rightarrow arcs; a; a = a \rightarrow next$ ) {
    if ( $get\_cap\_residual(a) > 0 \wedge a \rightarrow tip \neq i$ ) {
      if ( $a \rightarrow tip \rightarrow dist < min \vee min \equiv -1$ )  $min = a \rightarrow tip \rightarrow dist$ ;
    }
  }
   $i \rightarrow dist = min + 1$ ;
   $remove\_fila()$ ;
   $insere\_fila(i)$ ;
```

Este código é usado no bloco 13.

16. Execução de um push

Executar um push resume-se a atualizar valores: o fluxo no arco ou em seu irmão deve ser atualizado, bem como os excessos em seus extremos. Ademais, se a ponta final do arco que sofre a operação torna-se ativo, então tal vértice deve entrar na fila.

```
⟨ Executa um push 16 ⟩ ≡
   $j = a \rightarrow tip$ ;
   $alpha = get\_cap\_residual(a)$ ;
  if ( $i \rightarrow exc < alpha$ ) {
     $alpha = i \rightarrow exc$ ;
  }
  if ( $\neg arco\_original(a)$ ) {
     $a \rightarrow irmao \rightarrow flux -= alpha$ ;
  }
  else {
     $a \rightarrow flux += alpha$ ;
  }
   $i \rightarrow exc -= alpha$ ;
   $j \rightarrow exc += alpha$ ;
  if ( $i \rightarrow exc \equiv 0$ ) {
     $remove\_fila()$ ;
  }
  if ( $j \neq sorvedouro \wedge j \rightarrow exc \equiv alpha$ ) {
     $insere\_fila(j)$ ;
  }
}
```

Este código é usado no bloco 13.

17. Resta, agora, declarar as variáveis da função.

⟨Variáveis da função *fila_de_ativos* 17⟩ ≡

int *iteracoes*, *min*;

long *alpha*;

Vertex **i*, **j*;

Arc **a*;

Este código é usado no bloco 5.

18. Fila de vértices ativos

A fila utilizada para ordenar os vértices ativos a serem processados é implementada como uma fila circular através da utilização do próprio vetor de vértices de um grafo do SGB.

O campo i -*vertice* de determinado vértice i do grafo g , utilizado para a construção da fila, representa o vértice que ocupa a mesma posição de i no vetor g -*vertices*. O primeiro vértice na fila é dado por ini -*vertices* e o último, por fim -*vertices*. A condição ($ini \equiv fim$) indica que a fila está vazia. A implementação supõe que no máximo g - n vértices ocuparão a fila ao mesmo tempo.

Esta implementação de fila também é utilizada para a busca em largura efetuada na rede para a inicialização dos rótulos-distância.

```
< Fila circular de vértices 18 > ≡
Vertex * ini, *fim, *zero;
Vertex * max;
void inicializa_fila(Graph * g)
{
    ini = g-vertices;
    fim = g-vertices;
    zero = g-vertices;
    max = g-vertices + g-n;
    return;
}
void insere_fila(Vertex * i)
{
    fim-vertice = i;
    fim++;
    if (fim > max) {
        fim = zero;
    }
    if (fim ≡ ini) {
        fprintf(stderr, "ERRO: Fila excedeu sua capacidade.\n");
        exit(-1);
    }
}
boolean fila_vazia()
{
    if (ini ≡ fim) return TRUE;
    return FALSE;
}
Vertex * remove_fila()
{
    Vertex * i;
    if (!fila_vazia()) {
        i = ini-vertice;
```

```

    ini++;
    if (ini > max) {
        ini = zero;
    }
    return i;
}
return  $\Lambda$ ;
}
Vertex * primeiro_fila()
{
    if ( $\neg$ fila_vazia()) {
        return ini-vertice;
    }
    return  $\Lambda$ ;
}

```

Este código é usado no bloco 28.

19. Função principal

O programa consiste de três fases: inicialização, execução do algoritmo e finalização. A inicialização consiste em verificar a consistência dos parâmetros de entrada. A aplicação do algoritmo é simplesmente a chamada da função que já definimos anteriormente. A finalização consiste em imprimir no arquivo de saída o fluxo máximo obtido e o separador de capacidade mínima.

```
< Função principal 19 > ≡
int main(int argc, char *argv[])
{
    Graph *g;
    Vertex *fonte, *sorvedouro;
    boolean distancia;
    < Variáveis secundárias da função principal 27 >
    < Verifica consistência dos parâmetros 20 >
    fila_de_ativos(g, fonte, sorvedouro, distancia);
    < Imprime fluxo máximo 25 >
    < Imprime separador de capacidade mínima 26 >
    return (0);
}
```

Este código é usado no bloco 28.

20. Consistência dos parâmetros

Para que o programa seja executado corretamente, exige-se que o nome de arquivo fornecido referencie um grafo válido no formato SGB, onde o campo *len* de cada arco corresponda à sua capacidade. Também é necessário que os nomes de vértices referenciem vértices que de fato existam no grafo e que a rede contenha somente capacidades não-negativas. Por fim, é preciso verificar se a opção *-d* foi informada, indicando que os rótulos devem ser inicializados com distâncias na rede. Caso um número de parâmetros incorreto seja fornecido, as instruções do programa são impressas, exibindo a sintaxe.

```
< Verifica consistência dos parâmetros 20 > ≡
if (argc ≠ 5 ∧ argc ≠ 6) {
    fprintf(stderr, "Uso: %s<in><out>\ "source\ "\ "sink\ "\ [-d]\ n",
           argv[0]);
    exit(-1);
}
< Verifica validade dos arquivos 21 >
< Verifica existência dos vértices 22 >
< Verifica sinal das capacidades 23 >
< Verifica opção de inicialização dos rótulos 24 >
```

Este código é usado no bloco 19.

21. O programa utiliza as funções padrão para abrir o arquivo desejado. Caso o arquivo não possa ser aberto, o programa é imediatamente interrompido.

```

< Verifica validade dos arquivos 21 > ≡
  if ((g = restore_graph(argv[1])) ≡ Λ) {
    fprintf(stderr, "ERRO: Problemas com arquivo de entrada.\n");
    exit(-2);
  }
  if ((saida = fopen(argv[2], "w")) ≡ Λ) {
    fprintf(stderr, "ERRO: Arquivo de saída inválido.\n");
    exit(-3);
  }

```

Este código é usado no bloco 20.

22. Os vértices do grafo são examinados um por um até que os nomes fornecidos sejam encontrados. No caso de nomes iguais, considera-se o primeiro.

```

< Verifica existência dos vértices 22 > ≡
  fonte = Λ;
  sorvedouro = Λ;
  for (i = g-vertices; i < g-vertices + g-n; i++) {
    if (!strcmp(i-name, argv[3])) fonte = i;
    if (!strcmp(i-name, argv[4])) sorvedouro = i;
  }
  if (fonte ≡ Λ ∨ sorvedouro ≡ Λ) {
    fprintf(stderr, "ERRO: Vértices inválidos.\n");
    exit(-4);
  }

```

Este código é usado no bloco 20.

23. Os arcos do grafo são examinados um por um. O programa é interrompido imediatamente se um arco com capacidade negativa for encontrado.

```

< Verifica sinal das capacidades 23 > ≡
  for (i = g-vertices; i < g-vertices + g-n; i++) {
    for (a = i-arcs; a; a = a-next) {
      if (a-cap < 0) {
        fprintf(stderr, "ERRO: Capacidade negativa encontrada.\n");
        exit(-5);
      }
    }
  }

```

Este código é usado no bloco 20.

24. A opção `-d` indica que os rótulos dos vértices devem ser inicializados com a distância entre cada vértice e o sorvedouro.

```

< Verifica opção de inicialização dos rótulos 24 > ≡
    distancia = FALSE;
    if (argc ≡ 6) {
        if (strcmp("-d", argv[5]) ≡ 0) distancia = TRUE;
        else {
            fprintf(stderr, "ERRO: opção inválida: %s.\n", argv[5]);
            exit(-6);
        }
    }

```

Este código é usado no bloco 20.

25. Impressão do fluxo de intensidade máxima

Após a execução do algoritmo, imprime-se o fluxo máximo encontrado e sua intensidade.

```

< Imprime fluxo máximo 25 > ≡
    for (max = 0, i = g-vertices; i < g-vertices + g-n; i++) {
        for (a = i-arcs; a; a = a-next) {
            if (arco_original(a)) {
                fprintf(saida, "Fluxo de %s\ %a\ %s\ ": %ld\n",
                    a-inicio-name, a-tip-name, a-flx);
                if (i ≡ sorvedouro) max -= a-flx;
                if (a-tip ≡ sorvedouro) max += a-flx;
            }
        }
    }
    fprintf(saida, "Intensidade: %d\n", max);
    fprintf(stdout, "Intensidade do fluxo máximo: %d\n", max);

```

Este código é usado no bloco 19.

26. Impressão do separador de capacidade mínima

O separador de capacidade mínima é definido por um valor k tal que $d(i) \neq k$ para todo vértice i . Os vértices do separador possuem distância maior que k .

```

< Imprime separador de capacidade mínima 26 > ≡
    for (k = 1; k < g-n; k++) {
        for (i = g-vertices; i < g-vertices + g-n; i++) {
            if (i-dist ≡ k) break;
        }
        if (i ≡ g-vertices + g-n) break;
    }
    fprintf(saida, "\nSeparador:\n");

```

```

for (min = 0, i = g→vertices; i < g→vertices + g→n; i++) {
    if (i→dist > k) {
        fprintf(saida, "\ "%s"\ "\n", i→name);
        for (a = i→arcs; a; a = a→next) {
            if (a→flx ≥ 0 ∧ a→tip→dist < k) min += a→cap;
        }
    }
}
fprintf(saida, "Capacidade:_%d\n", min);
fprintf(stdout, "Capacidade_do_separador_minimo:_%d\n", min);
fclose(saida);

```

Este código é usado no bloco 19.

27. Podemos, agora, definir as variáveis secundárias da função principal.

⟨ Variáveis secundárias da função principal 27 ⟩ ≡

```

    Vertex * i;
    Arc * a;
    int min, max, k;
    FILE *saida;

```

Este código é usado no bloco 19.

28. Estrutura geral

Para concluir o programa, basta definir a estrutura geral.

```
< Bibliotecas necessárias 29 >  
< Fila circular de vértices 18 >  
< Função arco_original 8 >  
< Função get_capacidade_residual 14 >  
< Algoritmo da fila de vértices ativos 5 >  
< Função principal 19 >
```

29. Bibliotecas

Além das bibliotecas básicas, é preciso utilizar a plataforma SGB.

```
< Bibliotecas necessárias 29 > ≡  
#include <stdio.h>  
#include <stdlib.h>  
#include <gb_graph.h>  
#include <gb_save.h>
```

Este código é usado no bloco 28.

30. Macros

Definimos aqui todas as macros utilizadas no programa.

```
#define boolean int  
#define FALSE 0  
#define TRUE 1  
#define dist u.I  
#define exc v.I  
#define atual w.A  
#define vertice x.V  
#define cap len  
#define flx a.I  
#define irmao b.A  
#define inicio irmao→tip
```

Índice Remissivo

alpha: 16, 17.
Arc: 8, 14, 17, 27.
arco_original: 7, 8, 10, 11, 14, 16, 25.
arcs: 6, 7, 10, 11, 13, 15, 23, 25, 26.
argc: 19, 20, 24.
argv: 19, 20, 21, 22, 24.
atual: 6, 13, 30.
boolean: 5, 18, 19, 30.
cap: 7, 10, 14, 23, 26, 30.
dist: 11, 13, 15, 26, 30.
distancia: 5, 11, 19, 24.
exc: 6, 10, 12, 16, 30.
exit: 18, 20, 21, 22, 23, 24.
FALSE: 18, 24, 30.
fclose: 26.
fila_de_ativos: 5, 19.
fila_vazia: 5, 11, 18.
fim: 18.
flx: 6, 7, 8, 10, 14, 16, 25, 26, 30.
fonte: 5, 10, 11, 19, 22.
fopen: 21.
fprintf: 5, 18, 20, 21, 22, 23, 24, 25, 26.
gb_new_arc: 7.
get_cap_residual: 13, 14, 15, 16.
Graph: 5, 18, 19.
ini: 18.
inicializa_fila: 11, 12, 18.
inicio: 25, 30.
insere_fila: 11, 12, 15, 16, 18.
irmao: 7, 14, 16, 30.
iteracoes: 5, 17.
k: 27.
len: 20, 30.
main: 19.
max: 18, 25, 27.
min: 15, 17, 26, 27.
name: 22, 25, 26.
next: 6, 7, 10, 11, 13, 15, 23, 25, 26.
primeiro_fila: 5, 18.
remove_fila: 11, 15, 16, 18.
restore_graph: 21.
saida: 21, 25, 26, 27.
sorvedouro: 5, 11, 12, 16, 19, 22, 25.
st: 5.
stderr: 18, 20, 21, 22, 23, 24.
stdout: 5, 25, 26.
strcmp: 22, 24.
tip: 7, 10, 11, 13, 15, 16, 25, 26, 30.
TRUE: 18, 24, 30.
Vertex: 5, 17, 18, 19, 27.
vertice: 18, 30.
vertices: 6, 7, 11, 12, 18, 22, 23, 25, 26.
zero: 18.

Lista de Refinamentos

- ⟨ Algoritmo da fila de vértices ativos 5 ⟩ Usado no bloco 28.
- ⟨ Bibliotecas necessárias 29 ⟩ Usado no bloco 28.
- ⟨ Constrói arcos irmãos 7 ⟩ Usado no bloco 5.
- ⟨ Define pré-fluxo inicial 10 ⟩ Usado no bloco 9.
- ⟨ Define rótulos distância 11 ⟩ Usado no bloco 9.
- ⟨ Executa pré-processamento 9 ⟩ Usado no bloco 5.
- ⟨ Executa push ou relabel 13 ⟩ Usado no bloco 5.
- ⟨ Executa um push 16 ⟩ Usado no bloco 13.
- ⟨ Executa um relabel 15 ⟩ Usado no bloco 13.
- ⟨ Fila circular de vértices 18 ⟩ Usado no bloco 28.
- ⟨ Função principal 19 ⟩ Usado no bloco 28.
- ⟨ Função *arco_original* 8 ⟩ Usado no bloco 28.
- ⟨ Função *get_capacidade_residual* 14 ⟩ Usado no bloco 28.
- ⟨ Imprime fluxo máximo 25 ⟩ Usado no bloco 19.
- ⟨ Imprime separador de capacidade mínima 26 ⟩ Usado no bloco 19.
- ⟨ Inicializa fluxo 6 ⟩ Usado no bloco 5.
- ⟨ Insere vértices iniciais na fila 12 ⟩ Usado no bloco 5.
- ⟨ Variáveis da função *fila_de_ativos* 17 ⟩ Usado no bloco 5.
- ⟨ Variáveis secundárias da função principal 27 ⟩ Usado no bloco 19.
- ⟨ Verifica consistência dos parâmetros 20 ⟩ Usado no bloco 19.
- ⟨ Verifica existência dos vértices 22 ⟩ Usado no bloco 20.
- ⟨ Verifica opção de inicialização dos rótulos 24 ⟩ Usado no bloco 20.
- ⟨ Verifica sinal das capacidades 23 ⟩ Usado no bloco 20.
- ⟨ Verifica validade dos arquivos 21 ⟩ Usado no bloco 20.