

# PHarr XCPC ex Templates

PHarr

SMU

November 3, 2024

# Contents

<b>编程技巧和基础算法</b>	<b>2</b>
Linux 下运行脚本 . . . . .	2
<b>数据结构</b>	<b>2</b>
<b>图论</b>	<b>2</b>
差分约束系统 . . . . .	2
<b>数学知识</b>	<b>3</b>
计算几何 . . . . .	3
博弈论 . . . . .	4
<b>字符串</b>	<b>5</b>
后缀自动机 . . . . .	5
<b>动态规划</b>	<b>5</b>

## 编程技巧和基础算法

### Linux 下运行脚本

将以下脚本保存为 `run.sh`，如果要编译则 `./run.sh A.cpp`。

如果遇到了权限不足的情况可以 `chmod +x ./run.sh` 或者使用 `sudo`

```
1 #!/bin/bash
2 g++ $1.cpp -o $1 -g -O2 -std=c++20 -Wl,--stack=268435456 \
3 -Wall -fsanitize=undefined -fsanitize=address \
4 && echo compile_successfully >&2 && ./ $1
```

以下是一个可以在 MAC OS 上使用的版本

```
1 #!/bin/zsh
2 g++-11 $1.cpp -o $1 -g -O2 -std=c++20 \
3 -Wall -fsanitize=undefined -fsanitize=address \
4 && echo compile_successfully >&2 && ./ $1
```

## 数据结构

### 图论

#### 差分约束系统

##### 定义

差分约束系统是一种特殊的  $n$  元一次不等式组，它包含  $n$  个变量  $x_1, x_2, \dots, x_n$  以及  $m$  个约束条件，每个约束条件是由两个其中的变量做差构成的，形如  $x_i - x_j \leq c_k$ ，其中  $1 \leq i, j \leq n, i \neq j, 1 \leq k \leq m$  并且  $c_k$  是常数（可以是非负数，也可以是负数）。我们要解决的问题是：求一组解  $x_1 = a_1, x_2 = a_2, \dots, x_n = a_n$ ，使得所有的约束条件得到满足，否则判断出无解。

差分约束系统中的每个约束条件  $x_i - x_j \leq c_k$  都可以变形为  $x_i \leq x_j + c_k$ ，这与单源最短路中的三角形不等式  $dist[y] \leq dist[x] + z$  非常相似。因此，我们可以把每个变量  $x_i$  看做图中的一个结点，对于每个约束条件  $x_i - x_j \leq c_k$ ，从结点  $j$  向结点  $i$  连一条长度为  $c_k$  的有向边。

注意到，如果  $\{a_1, a_2, \dots, a_n\}$  是该差分约束系统的一组解，那么对于任意的常数  $d$ ， $\{a_1 + d, a_2 + d, \dots, a_n + d\}$  显然也是该差分约束系统的一组解，因为这样做差后  $d$  刚好被消掉。

##### 过程

设  $dist[0] = 0$  并向每一个点连一条权重为 0 边，跑单源最短路，若图中存在负环，则给定的差分约束系统无解，否则， $x_i = dist[i]$  为该差分约束系统的一组解。

##### 性质

一般使用 Bellman-Ford 或队列优化的 Bellman-Ford（俗称 SPFA，在某些随机图跑得很快）判断图中是否存在负环，最坏时间复杂度为  $O(nm)$ 。

如果题目给定了一个源点，则不需要建立超级源点。

```
1 // luogu P1993
2 // 有三种约束条件
3 // x[a] >= x[b] + c -> x[b] - x[a] <= -c -> add(a, b, -c)
4 // x[a] <= x[b] + c -> x[a] - x[b] <= c -> add(b, a, c)
5 // x[a] == x[b] -> x[a] - x[b] <= 0 and x[b] - x[a] <= 0 -> add(a, b, 0), add(b, a, 0)
6 #include <bits/stdc++.h>
7
8 using namespace std;
9
10 const int inf = INT_MAX / 2;
11
12 using vi = vector<int>;
13 using pii = pair<int, int>;
14
15 int main() {
16     ios::sync_with_stdio(false), cin.tie(nullptr);
17     int n, m;
```

```

18     cin >> n >> m;
19
20     vector<vector<pii>> e(n + 1);
21     for (int op, a, b, c; m; m--) {
22         cin >> op;
23         if (op == 1) {
24             cin >> a >> b >> c;
25             e[a].emplace_back(b, -c);
26         } else if (op == 2) {
27             cin >> a >> b >> c;
28             e[b].emplace_back(a, c);
29         } else {
30             cin >> a >> b;
31             e[a].emplace_back(b, 0);
32             e[b].emplace_back(a, 0);
33         }
34     }
35
36     for (int i = 1; i <= n; i++)
37         e[0].emplace_back(i, 0);
38
39     vector<int> dis(n + 1, inf), vis(n + 1), tot(n + 1);
40     dis[0] = 0, vis[0] = 1, tot[0]++;
41     bool ok = true;
42     queue<int> q;
43     q.push(0);
44     while (not q.empty() and ok) {
45         int x = q.front();
46         q.pop();
47         vis[x] = 0;
48         for (auto [y, w]: e[x]) {
49             if (dis[y] <= dis[x] + w) continue;
50             dis[y] = dis[x] + w;
51             if (vis[y] != 0) continue;
52             vis[y] = 1;
53             q.push(y);
54             tot[y]++;
55             if (tot[y] > n) {
56                 ok = false;
57                 break;
58             }
59         }
60     }
61
62     if (ok) cout << "Yes\n";
63     else cout << "No\n";
64     return 0;
65 }
66

```

## 数学知识

### 计算几何

已知正方形对角线两点坐标，求另外两点坐标

按照顺时针方向，正方形上四点  $A, B, C, D$ ，两对角线交点  $O$ 。现在已知  $A(ax, ay), C(cx, cy)$  求另外两点坐标。

令  $\vec{v} = \frac{C-A}{2} = (\frac{cx-ax}{2}, \frac{cy-ay}{2})$ ，则有  $O = A + \vec{v} = C - \vec{v} = (ax + vx, ay + vy) = (cx - vx, cy - vy)$

根据正方形的对称性可知

$$B = (ox - vy, oy + vx) = (ax + vx - vy, cy + vx - vy) D = (ox + vy, oy - vx) = (cx - vx + vy, ay - vx + vy)$$

令  $vp = vx - vy = \frac{cx-ax-cy+ay}{2}$  分别代入上式子可得

$$B = (ax + vp, cy + vp) C = (cx - vp, ay - vp)$$

## 博弈论

### Lasker's Nim Game

$n$  堆石子，每次玩家可以从一堆石子中取走若干个石子，或者把一堆石子分成两个非空的堆

考虑暴力的求解每一堆石子的 SG 函数

$$SG(x) = \begin{cases} 0 & x = 0 \\ \text{mex}\{SG(0)\} = 1 & x = 1 \\ \text{mex}\{SG(x-1), SG(x-2), \dots, \text{mex}\{SG(y) \oplus SG(z) | (y > 0 \wedge z > 0 \wedge y+z = x)\}\} & x \geq 2 \end{cases}$$

然后我们打表找规律可以得到

$$SG(x) = \begin{cases} 0 & \text{if } x = 0 \\ x & \text{if } x = 4k+1 \vee x = 4k+2 \\ x \oplus 1 & \text{if } x = 4k+3 \vee x = 4k+4 \end{cases}$$

### 移动棋子

有一个  $1 \times n$  的棋盘，其中每个格子可以有多个棋子，每次可以选择一个棋子，将其移动到更左边的任意一个格子，两个人轮流移动，不能移动则输。

考虑每一个棋子的 SG 函数。一个棋子如果在从左向右第  $i (i \geq 0)$  个格子，则  $SG(i) = i$ 。

### [HNOI2007] 分裂游戏

有  $n$  堆石子，每堆有  $a_i$  个石子，保证  $0 \leq n \leq 21, 0 \leq a_i \leq 10^4$ 。两个玩家轮流操作，每次可以从第  $i$  堆拿出一个石子，并在  $j, k (i < j \leq k \leq n)$  堆中各放入一个石子。不能操作的人输。

求出每一个石子的 SG 函数，一个在位置  $i$  的石子  $SG(i) = \text{mex}\{SG(l) \oplus SG(r) | i < j \leq k \leq n\}$ 。可以  $O(N^3)$  预处理每一个石子的 SG 函数。

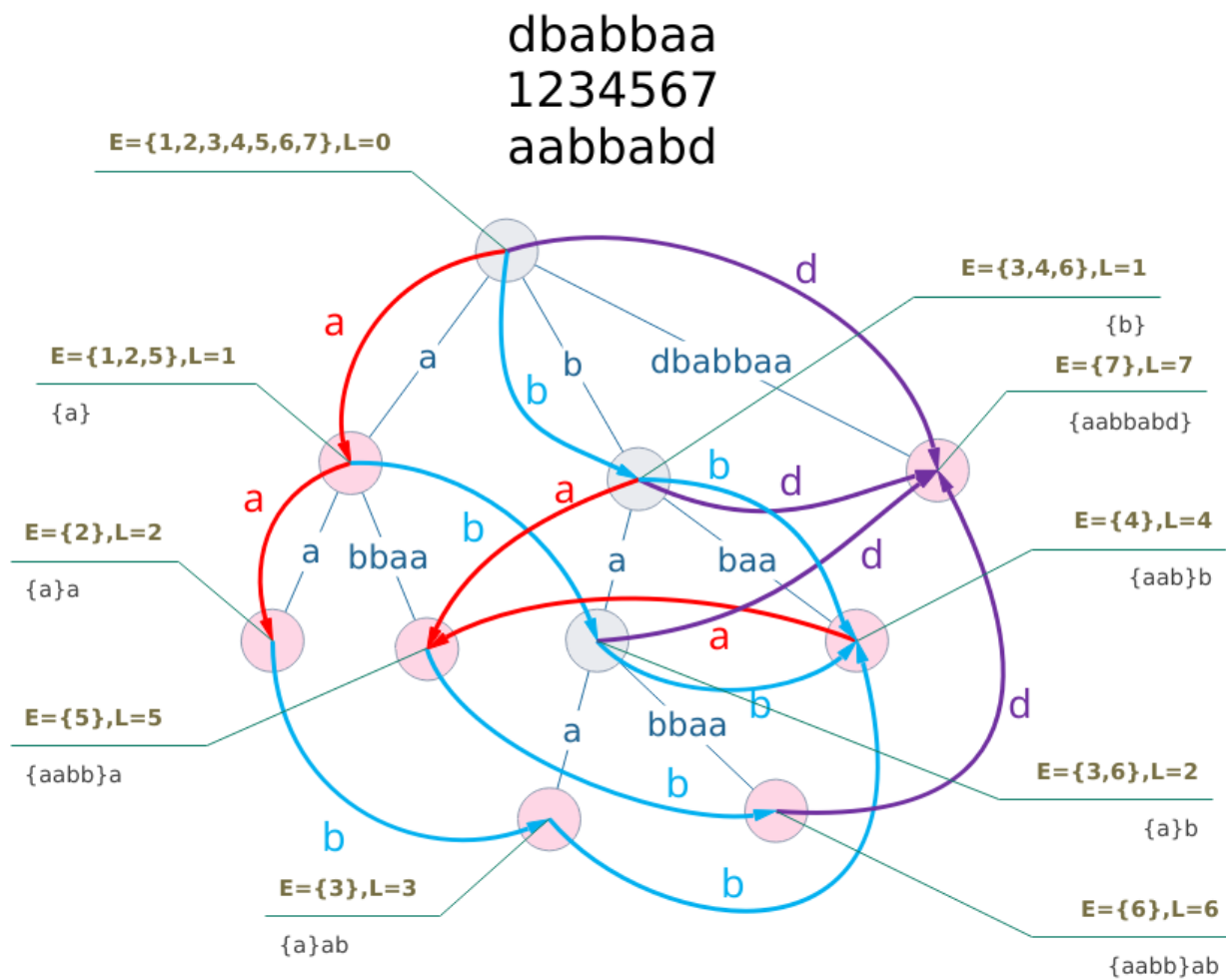
### Green Hackenbush Game on Tree(树上删边游戏)

给一个有根树（森林），每次可以删掉一个子树。

叶子点的 SG 值为 0，非叶子点的  $SG(u) = \oplus [SG(v) + 1]$ ， $v$  是  $u$  的子节点。

## 字符串

### 后缀自动机



### 动态规划