UBND TỈNH NINH BÌNH
TRƯỜNG ĐẠI HỌC HOA LƯ

BÁO CÁO KẾT QUẢ THỰC HIỆN NHIỆM VỤ KHOA HỌC VÀ CÔNG NGHỆ CẤP CƠ SỞ

PHƯƠNG PHÁP QUY HOẠCH ĐỘNG VÀ MỘT SỐ BÀI TOÁN BỒI DƯỠNG HỌC SINH GIỎI TIN HỌC PHỔ THÔNG

Chủ nhiệm: **ThS. PHÙNG THỊ THAO**
Đơn vị công tác: **TRƯỜNG PTTHSP TRÀNG AN**

NINH BÌNH, 2022

UBND TỈNH NINH BÌNH

TRƯỜNG ĐẠI HỌC HOA LƯ

BÁO CÁO KẾT QUẢ THỰC HIỆN NHIỆM VỤ KHOA HỌC VÀ CÔNG NGHỆ CẤP CƠ SỞ

PHƯƠNG PHÁP QUY HOẠCH ĐỘNG VÀ MỘT SỐ BÀI TOÁN BỒI DƯỠNG HỌC SINH GIỎI TIN HỌC PHỔ THÔNG

Chủ nhiệm: **ThS. Phùng Thị Thao**

Đơn vị: Trường PTTHSP Tràng An

Các thành viên: **ThS. Lã Đăng Hiệp**

Đơn vị: Phòng KT & ĐBCL

Xác nhận của Chủ tịch HĐ nghiệm thu **Chủ nhiệm nhiệm vụ**

TS. Lâm Văn Năng **ThS. Phùng Thị Thao**

NINH BÌNH, 2022

MỤC LỤC

THÔNG TIN KẾT QUẢ NGHIÊN CỨU ii
MỞ ĐẦU iii
CHƯƠNG 1. PHƯƠNG PHÁP QUY HOẠCH ĐỘNG VÀ CÁC BÀI TOÁN ĐIỂN HÌNH 1
1.1. Các khái niệm trong Phương pháp quy hoạch động 1
1.2. Phương pháp giải bài toán quy hoạch động 3
1.3. Phân tích một số bài toán điển hình 5
1.3.1. Tìm số Fibonaci thứ n 5
1.3.2. Xâu con chung dài nhất 6
1.3.3. Tìm dãy con tăng dài nhất 7
1.3.4. Bài toán cái túi 9
1.3.5. Bài toán di chuyển từ Tây sang Đông 10
CHƯƠNG 2. MỘT SỐ BÀI TẬP ỨNG DỤNG QUY HOẠCH ĐỘNG GIẢI BẰNG PHƯƠNG PHÁP PHÂN TÍCH DỮ LIỆU CUỐI 12
2.1. Một số bài tập ứng dụng quy hoạch động 12
2.1.1. Phương pháp đơn vị dữ liệu cuối 12
2.2.2. Cơ sở toán học 12
2.2.3. Các bài toán ứng dụng phương pháp quy hoạch động 13
2.2. Xây dựng chương trình và bộ test để kiểm tra tính tối ưu giải thuật của một số bài toán 25
2.2.1. Bài Toán 1: ĐẾM XÂU NHỊ PHÂN 25
2.2.2. Bài Toán 2: ĐẾM XÂU NHỊ PHÂN MỞ RỘNG 25
2.2.3. Bài Toán 3: LÁT VIỀN 26
2.2.4. Bài toán 4: ĐẾM SỐ DÃY CON TĂNG DÀI NHẤT 27
2.2.5. Bài toán 5: CẶP SỐ 0 28
2.2.6. Bài toán 6: BỜM UỐNG RƯỢU 29
2.2.7. Bài toán 7: TRÒ CHƠI VỚI BĂNG SỐ 29
2.2.8. Bài toán 8: NỐI MẠNG MÁY TÍNH. 30
2.2.9. Bài toán 9: NHẢY VỀ ĐÍCH (Bài toán mở rộng của bài toán 1.3.5) 31
2.2.10. Bài toán 10: LÒ CÒ 32
KẾT LUẬN VÀ KIẾN NGHỊ 33
TÀI LIỆU THAM KHẢO 34

THÔNG TIN KẾT QUẢ NGHIÊN CỨU

1. Giới thiệu vấn đề nghiên cứu và mục tiêu nghiên cứu

Đề tài nghiên cứu việc giải quyết một lớp bài toán tối ưu bằng phương pháp quy hoạch động. Mục tiêu nghiên cứu phân loại lớp bài toán này theo một số dạng toán cụ thể, lập trình giải quyết bài toán và sinh test tự động bằng ngôn ngữ lập trình C++.

2. Tính mới và tính sáng tạo của nghiên cứu

Đề tài đã phân nhóm thành 3 dạng toán cụ thể; Lập trình giải quyết các bài toán bằng ngôn ngữ lập trình C++, viết mã chương trình (code) để sinh tự động dữ liệu vào và dữ liệu ra (test) có giá trị lớn, tính toán độ phức tạp của giải thuật của các bài toán đặt ra để thấy được tính ưu việt của phương pháp quy hoạch động

3. Tóm lược kết quả nghiên cứu.

Đề tài đã nghiên cứu lý thuyết về phương pháp quy hoạch động và nghiên cứu sâu về phương pháp giải bài toán quy hoạch động bằng cách sử dụng đơn vị dữ liệu cuối, xây dựng chương trình, sinh bộ test tự động bằng ngôn ngữ lập trình C++ cho một số bài tập từ điển hình đến các bài tập mở rộng, xác định độ phức tạp của mỗi giải thuật được đưa ra.

4. Đóng góp về mặt kinh tế - xã hội, giáo dục và đào tạo, an ninh quốc phòng và khả năng áp dụng của nghiên cứu

Đề tài góp phần nâng cao năng lực của nhóm nghiên cứu, hỗ trợ giáo viên trong việc giảng dạy ôn luyện thi học sinh giỏi môn tin học Trung học phổ thông cấp tỉnh.

MỞ ĐẦU

1. Tổng quan tình hình nghiên cứu

Quy hoạch động (Dynamic Programming) là một phương pháp rất hiệu quả để giải nhiều bài toán tin học, đặc biệt là những bài toán tối ưu, có một số bài toán sử dụng phương pháp quy hoạch động lại cho hiệu quả cao hơn hơn hẳn so với nhiều phương pháp khác. Các công trình nghiên cứu trong nước về phương pháp này được trình bày chủ yếu trong các cuốn sách chuyên ngành gồm cuốn "Tài liệu giáo khoa chuyên tin quyển 1" tác giả Hồ Sỹ Đàm (chủ biên) – NXB giáo dục Việt Nam 2009 [1]; và cuốn Bài giảng chuyên đề "Giải thuật và lập trình" dạng Ebook của TS. Lê Minh Hoàng – ĐH sư phạm Hà Nội năm 2002 [2], tổng hợp các phương pháp giải thuật cơ bản và chuyên sâu trong đó có phương pháp quy hoạch động. Có nhiều tài liệu đã trình bày phương pháp quy hoạch động và ứng dụng vào giải một số bài toán Tin học nổi tiếng, tuy nhiên số lượng bài toán tin học được giải bằng phương pháp quy hoạch động cũng rất lớn và luôn được cải tiến.

Đề tài nghiên cứu theo hướng xây dựng một số bài tập cụ thể có thể giải bằng phương pháp phân tích đơn vị dữ liệu cuối, xây dựng code và bộ test đầy đủ để kiểm tra tính đúng đắn, tối ưu của các bài toán đưa ra, đề tài có thể áp dụng trực tiếp vào giảng dạy trong chương trình bồi dưỡng học sinh giỏi của trường PTTHSP Tràng An, bồi dưỡng HSG các cấp, đặc biệt từ cấp tỉnh trở lên.

2. Tính cấp thiết của đề tài

Để giải các bài toán trong Tin học, việc xác định Thuật toán để giải bài toán là công việc đặc biệt quan trọng. Trong các bài toán tin học, có một lớp bài toán được gọi chung là bài toán tối ưu (bài toán có nhiều nghiệm mà ta phải đi tìm nghiệm tối ưu theo yêu cầu). Với lớp bài toán này có một số phương pháp giải có thể kể đến như: phương pháp duyệt – vét cạn, tham lam, nhánh cận và quy hoạch động (QHĐ). Với mỗi phương pháp đều có ưu, nhược điểm khác nhau, trong đó QHĐ là phương pháp được cho là hiệu quả nhất (xét về khả năng tìm ra nghiệm đúng và thời gian, cũng như bộ nhớ cần phải bỏ ra để giải quyết được bài toán).

Trong các đề thi HSG Tin học các cấp, có một số bài toán tối ưu cần sử dụng những chiến lược trên. Mặc dù, cũng có những cách khác để giải bài toán đó nhưng vì các đề thi đều có giới hạn về thời gian, cũng như bộ nhớ của

chương trình, nên một thuật toán hiệu quả là cực kỳ cần thiết. Và trong những trường hợp như vậy, quy hoạch động là một lựa chọn tốt.

Do đó, chúng tôi lựa chọn đề tài: ***Phương pháp quy hoạch động và một số bài toán bồi dưỡng học sinh giỏi tin học phổ thông.***

3. Mục tiêu nghiên cứu

Nghiên cứu phương pháp quy hoạch động thông qua việc giải một số bài toán quy hoạch động điển hình.

Xây dựng một số bài tập có thể giải bằng phương pháp phân tích đơn vị dữ liệu cuối, đóng góp thêm vào ngân hàng bài tập giúp hỗ trợ cho công tác bồi dưỡng học sinh giỏi môn Tin học.

4. Đối tượng nghiên cứu

- Cấu trúc dữ liệu và giải thuật.
- Các phương pháp thiết kế thuật toán.
- Phương pháp quy hoạch động.

5. Phạm vi nghiên cứu

- Nghiên cứu giải các bài toán quy hoạch động bằng cách phân tích đơn vị dữ liệu cuối.
- Ngôn ngữ lập trình C++ trong giải quyết các bài toán lập trình tin học cấp phổ thông.

6. Cách tiếp cận và phương pháp nghiên cứu

6.1. Cách tiếp cận

Tiếp cận từ lý thuyết → Tìm hiểu bài toán → xây dựng thuật toán → Lập trình giải quyết bài toán

6.1. Phương pháp nghiên cứu

- *Phương pháp nghiên cứu lý thuyết:* Nghiên cứu về các giải thuật.
- *Phương pháp thực nghiệm khoa học:* Lập trình giải bài toán và xây dựng bộ test để kiểm tra tính tối ưu của bài toán

CHƯƠNG 1. PHƯƠNG PHÁP QUY HOẠCH ĐỘNG VÀ CÁC BÀI TOÁN ĐIỂN HÌNH

1.1. Các khái niệm trong Phương pháp quy hoạch động

Phương pháp quy hoạch động dùng để giải bài toán tối ưu có bản chất đệ quy, tức là việc tìm phương án tối ưu cho bài toán đó có thể đưa về tìm phương án tối ưu của một số hữu hạn các bài toán con. Đối với nhiều thuật toán đệ quy chúng ta đã tìm hiểu, nguyên lý chia để trị (divide and conquer) thường đóng vai trò chủ đạo trong việc thiết kế thuật toán. Để giải quyết một bài toán lớn, ta chia nó làm nhiều bài toán con cùng dạng với nó để có thể giải quyết độc lập. Trong phương pháp quy hoạch động, nguyên lý này càng được thể hiện rõ: Khi không biết cần phải giải quyết những bài toán con nào, ta sẽ đi **giải quyết tất cả các bài toán con** và **lưu trữ những lời giải hay đáp số của chúng** với mục đích **sử dụng lại** theo một sự phối hợp nào đó để giải quyết những bài toán tổng quát hơn. Đó chính là điểm khác nhau giữa Quy hoạch động và phép phân giải đệ quy và cũng là nội dung phương pháp quy hoạch động:

Phép phân giải đệ quy **bắt đầu từ bài toán lớn** phân rã thành nhiều bài toán con và đi giải từng bài toán con đó. Việc giải từng bài toán con lại đưa về phép phân rã tiếp thành nhiều bài toán nhỏ hơn và lại đi giải tiếp bài toán nhỏ hơn đó bất kể nó đã được giải hay chưa [1].

Quy hoạch động bắt đầu từ việc **giải tất cả các bài toán nhỏ nhấ**t (bài toán cơ sở) để từ đó từng bước giải quyết những bài toán lớn hơn, cho tới khi giải được bài toán lớn nhất (bài toán ban đầu).

Ta xét một ví dụ đơn giản:

Ví dụ: Dãy Fibonacci là dãy số nguyên dương được định nghĩa như sau:

$F_1 = F_2 = 1$;

$F_i = F_{i-1} + F_{i-2}$ với mọi $i \geq 3$

Hãy tính F_n

Xét hai cách cài đặt chương trình:

Cách 1

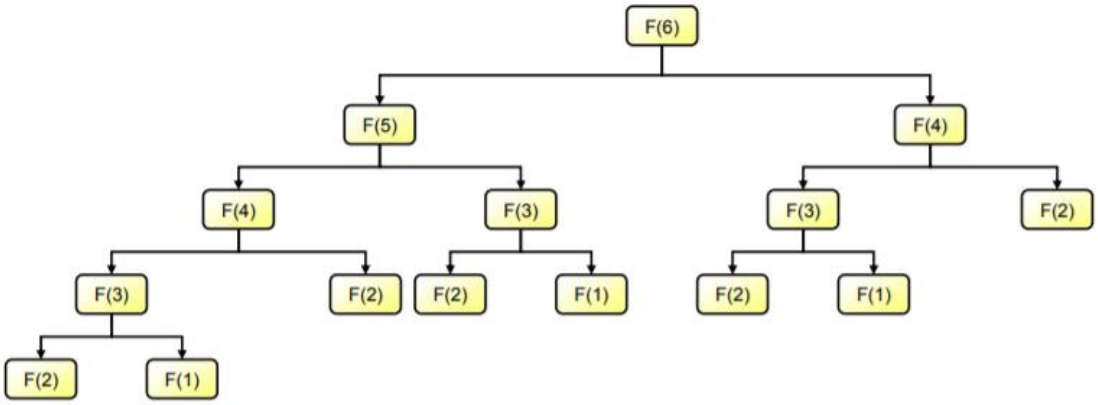
```
int F(int i)
{
  int res=0;
  if (i < 3) res= 1;
```

```
    else res = F(i - 1) + F(i - 2);
    return res;
}
```

Cách 2

```
Int fb2(int n)
{
  f[1] = f[2] =1;
  for(int i=3; i<=n; i++) f[i] = f[i-1] +f[i-2];
 }
```

Trong cách 1, ta viết một hàm đệ quy F(i) để tính số Fibonacci thứ i. Chương trình chính gọi F(6), nó sẽ gọi tiếp F(5) và F(4) để tính … Quá trình tính toán có thể vẽ như cây dưới đây. Ta nhận thấy để tính F(6) nó phải tính 1 lần F(5), hai lần F(4), ba lần F(3), năm lần F(2), ba lần F(1).



Hàm đệ quy tính số Fibonacci

Cách 2 thì không như vậy. Trước hết nó tính sẵn F[1] và F[2], từ đó tính tiếp F[3], lại tính tiếp được F[4], F[5], F[6]. Đảm bảo rằng mỗi giá trị Fibonacci chỉ phải tính 1 lần.

Trước khi áp dụng phương pháp quy hoạch động ta phải xét xem phương pháp đó có thoả mãn những yêu cầu dưới đây hay không:

Bài toán lớn phải phân rã được thành nhiều bài toán con, mà sự phối hợp lời giải của các bài toán con đó cho ta lời giải của bài toán lớn.

Vì quy hoạch động là đi giải tất cả các bài toán con, nên nếu không đủ không gian vật lý lưu trữ lời giải (bộ nhớ, đĩa…) để phối hợp chúng thì phương pháp quy hoạch động cũng không thể thực hiện được.

Quá trình từ bài toán cơ sở tìm ra lời giải bài toán ban đầu phải qua hữu hạn bước.

*** Các khái niệm** [1]:

- Bài toán giải theo phương pháp quy hoạch động gọi là **bài toán quy hoạch động**

- Công thức phối hợp nghiệm của các bài toán con để có nghiệm của bài toán lớn gọi là **công thức truy hồi** của quy hoạch động.

- Tập các bài toán nhỏ nhất có ngay lời giải để từ đó giải quyết các bài toán lớn hơn gọi là **cơ sở quy hoạch động.**

- Không gian lưu trữ lời giải các bài toán con để tìm cách phối hợp chúng gọi là **bảng phương án của quy hoạch động**

Các bước cài đặt một chương trình sử dụng quy hoạch động:

B1. Giải tất cả các bài toán cơ sở (thông thường rất dễ), lưu các lời giải vào bảng phương án.

B2. Dùng công thức truy hồi phối hợp những lời giải của những bài toán nhỏ đã lưu trong bảng phương án để tìm lời giải của những bài toán lớn hơn và lưu chúng vào bảng phương án. Cho tới khi bài toán ban đầu tìm được lời giải.

B3. Dựa vào bảng phương án, truy vết tìm ra nghiệm tối ưu.

Cho đến nay, vẫn chưa có một định lý nào cho biết một cách chính xác những bài toán nào có thể giải quyết hiệu quả bằng quy hoạch động. Tuy nhiên để biết được bài toán có thể giải bằng quy hoạch động hay không, ta có thể tự đặt câu hỏi: "***Một nghiệm tối ưu của bài toán lớn có phải là sự phối hợp các nghiệm tối ưu của các bài toán con hay không?"*** và ***"Liệu có thể nào lưu trữ được nghiệm các bài toán con dưới một hình thức nào đó để phối hợp tìm được nghiệm bài toán lớn?"***. Nếu câu trả lời là có thì bài toán đó hoàn toàn có thể giải bằng phương pháp quy hoạch động [2].

1.2. Phương pháp giải bài toán quy hoạch động

Chúng ta dùng thuật toán quy hoạch động khi gặp:

- Bài toán có các bài toán con gối nhau.

- Bài toán có cấu trúc con tối ưu.

*** Bài toán con gối nhau**

Tương tự như thuật toán chia để trị, quy hoạch động cũng chia bài toán lớn thành các bài toán con nhỏ hơn. Quy hoạch động được sử dụng khi các bài toán con này được gọi đi gọi lại. Phương pháp quy hoạch động sẽ lưu kết

quả của bài toán con này, và khi được gọi, nó sẽ không cần phải tính lại, do đó làm giảm thời gian tính toán.

Quy hoạch động sẽ không thể áp dụng được (hoặc nói đúng hơn là áp dụng cũng không có tác dụng gì) khi các bài toán con không gối nhau. Ví dụ với thuật toán tìm kiếm nhị phân, quy hoạch động cũng không thể tối ưu được gì cả, bởi vì mỗi khi chia nhỏ bài toán lớn thành các bài toán con, mỗi bài toán cũng chỉ cần giải một lần mà không bao giờ được gọi lại. Một ví dụ điển hình của bài toán con gối nhau là bài toán tính số Fibonacci.

*** Cấu trúc con tối ưu**

Cấu trúc con tối ưu có tính chất là lời giải của bài toán lớn sẽ là tập hợp lời giải từ các bài toán nhỏ hơn.

Ví dụ: Trong bài toán tìm đường đi ngắn nhất trong đồ thị, nếu một node x nằm trên đường đi ngắn nhất giữa hai node u, v thì đường đi ngắn nhất từ u đến v sẽ là tổng hợp của đường đi ngắn nhất từ u đến x và đường đi ngắn nhất từ x đến v. Một số thuật toán tìm đường trên đồ thị (ví dụ thuật toán Dijkstra) đều dựa trên tính chất này, và nó được áp dụng trong quy hoạch động.

Tính chất cấu trúc con tối ưu rất quan trọng, nó cho phép chúng ta giải bài toán lớn dựa vào các bài toán con đã giải được. Nếu không có tính chất này, chúng ta không thể áp dụng quy hoạch động được.

*** Cách tiếp cận: Quy hoạch động xuôi và ngược** [3].

Ta có khái niệm sử dụng quy hoạch động kiểu "ngược". Ngược ở đây không phải là chúng ta duyệt các bài toán con từ lớn ngược về nhỏ. Mà quy trình là: Duyệt qua tất cả các bài toán con (từ nhỏ đến lớn), với mỗi bài toán đó, chúng ta tính toán kết quả dựa vào bài toán con trước đó. Tất nhiên, bài toán con phía trước đã được giải theo quy trình duyệt, và với mỗi bài toán, chúng ta phải "**nhìn ngược lại**" bài toán trước đó, nên cách làm này gọi là quy hoạch động kiểu "ngược".

Phương pháp quy hoạch động ngược này được sử dụng rộng rãi, vì nó khá tương ứng với suy nghĩ tự nhiên con người. Chúng ta đọc đề bài, suy nghĩ cách giải, cách giải đó yêu cầu phải giải những bài toán nhỏ hơn. Chúng ta tiếp tục suy nghĩ cho những bài toán con này, sau đó tổng hợp để tìm ra lời giải cho bài toán lớn. Quá trình cứ tiếp tục như vậy, và quy hoạch động kiểu "ngược" đang được xây dựng đúng theo trình tự đó.

Ngoài ra, về mặt lập trình, kiểu quy hoạch động này có mối quan hệ tương đối gần gũi với đệ quy. Một bài toán lớn được giải dựa vào các bài toán con tương tự nhau (và tương tự bài toán lớn) thì việc áp dụng đệ quy có thể là một phương pháp dễ dàng để code. Vì vậy, nhiều trường hợp, có thể coi quy hoạch động là một cách để tối ưu phương pháp đệ quy để giải một bài toán.

Ngoài kiểu quy hoạch động "ngược", có một kiểu quy hoạch động "xuôi". Tuy không phổ biến, kiểu quy hoạch động xuôi cũng khá khó áp dụng, nhưng quy hoạch động "xuôi" mang đến cho chúng ta nhiều tiện lợi. Với phương pháp xuôi này cũng cần duyệt qua các bài toán con từ nhỏ đến lớn, nhưng với mỗi bài toán con, chúng ta tính toán kết quả và từ đó tìm cách thực hiện một số phép tính để giải bài toán lớn hơn. Nghĩa là, với mỗi bài toán con, chúng ta sẽ **nhìn về phía trước** để xem phải giải bài toán tiếp theo như thế nào từ bài toán hiện tại.

Phương pháp này khó áp dụng hơn phương pháp ngược, và cũng không phải bài toán nào cũng áp dụng được. Với mỗi bài toán, việc xác định bước tiếp theo tương đối khó khăn, thậm chí việc kiểm tra tính đúng sai của phương pháp cũng không hề dễ dàng.

Thông thường, mỗi bài toán cần phải giải bằng cách tổng hợp kết quả từ một vài bài toán con trước đó. Vì vậy, cách quy hoạch động xuôi này chỉ sử dụng một bài toán con để tính toán trước bài toán tiếp theo sẽ chỉ cho ra một phần của kết quả chứ không phải kết quả cuối cùng. Vì vậy, để thực hiện quy hoạch động xuôi, việc điền sẵn một mảng các giá trị trung tính là điều bắt buộc (sau đó chúng ta sẽ cộng dồn kết quả vào mỗi khi giải được một bài toán con mới).

Sau đây ta sẽ đi vào một số bài toán quy hoạch động điển hình để xem xét cách quy hoạch động được áp dụng vào các bài toán cụ thể như thế nào.

1.3. Phân tích một số bài toán điển hình

1.3.1. Tìm số Fibonaci thứ n

Bài toán: Tìm số Fibonaci thứ n

Dữ liệu: Một số nguyên dương n

Kết quả: Ghi ra giá trị số Fibonaci thứ n

Như đã trình bày ở **1.1**, bài toán Fibonaci nếu giải theo phương pháp đệ quy thì độ phức tạp tính toán khá cao (có thể lên đến 2^{n-1}) và với độ lớn của n từ hơn 100 thì việc tính toán mất nhiều thời gian mới ra kết quả và không thể đáp ứng yêu cầu về mặt tốc độ tính toán, bộ nhớ lưu trữ.

Đặc trưng của bài toán là các bài toán con gối nhau, muốn tìm f_n ta phải tìm f_{n-1} và f_{n-2}, muốn tìm f_{n-1} ta phải tìm f_{n-2} và f_{n-3} cứ như vậy đến f_0

Giải:

Ta sẽ xác định được:

- Bài toán cơ sở là $f_0 = f_1 = 1$;
- Công thức truy hồi: $f_i = f_{i-1} + f_{i-2}$
- Bảng phương án ta dùng một mảng F[] để lưu các giá trị
- Nghiệm của bài toán chính là F[n]

Cài đặt:

```
#include <bits/stdc++.h>
#define maxn 100
using namespace std;
int f[maxn], n;
int main()
{
    cin>>n;
    f[0]=f[1]=1;
    for(int i=2; i<=n; i++) f[i]=f[i-1]+f[i-2];
    cout<<f[n];
    return 0;
}
```

1.3.2. Xâu con chung dài nhất

Bài toán: Cho hai xâu kí tự X và Y gồm các ký tự từ a đến z trong bảng chữ, hãy tìm xâu con chung lớn nhất Z gồm các kí tự thuộc vào cả 2 xâu X và Y giữ nguyên trật tự trước sau. Đưa ra độ dài của xâu con chung dài nhất.

Dữ liệu: gồm 2 dòng, dòng đầu là xâu kí tự X, dòng thứ 2 là xâu kí tự Y

Kết quả: Đưa ra kết quả bài toán

Giải:

Đặc trưng bài toán đó là: Giả sử phải tìm xâu con chung tại bước thứ k ta dựa vào kết quả tại bước thứ k-1, kết quả tại bước thứ k-1 dựa vào kết quả bước thứ k-2 tiếp tục cho tới bước đầu tiên

Xác định Hàm mục tiêu: Ta nhận thấy độ dài lớn nhất của xâu con chung phụ thuộc vào số kí tự đang xét của xâu X và số kí tự đang xét của xâu Y, như vậy hàm mục tiêu của ta sẽ có hai tham số.

Gọi L[i,j] là độ dài xâu con chung lớn nhất của X và Y khi xét đến i kí tự đầu của xâu X và j kí tự đầu của xâu Y.

Ta thấy rằng nếu một trong 2 xâu là rỗng thì độ dài xâu con chung bằng 0 nghĩa là L[0,j]=L[i,0]=0; (Bài toán cơ sở)

Nếu X[i]= Y[j] thì độ dài L[i,j] tăng lên 1; L[i,j]=L[i-1,j-1] +1;

Còn trái lại ta phải xét xem X[i] có bằng Y[j-1] và Y[j] có bằng X[i-1] không? Nghĩa là phải xét đến L[i,j-1] và L[i-1,j] xem giá trị nào lớn hơn thì chấp nhận: L[i,j]= Max(L[i,j-1] , L[i-1,j])

Ta có công thức truy hồi sau

$$L_{i,j} = \begin{cases} L_{i-1,j-1} + 1 & khi \quad X_i = Y_j \\ Max(L_{i-1,j}, L_{i,j-1}) \, khi & X_i \neq Y_j \end{cases}$$

Dùng mảng L[][] để lưu bảng phương án

Kết quả bài toán là giá trị lớn nhất của L[i,j] (i=1..n và j=1..m)

Cài đặt:

```
#include <bits/stdc++.h>
#define maxn 1000
using namespace std;
string x,y;
int l[maxn][maxn]; int n,m; int res=0;
int main()
{
    cin>>x>>y;
    n=x.size(); m=y.size();
    for(int i=0; i<n; i++) l[i][0] = 0;
    for(int j=0; j<m; j++) l[0][j] = 0;
    for(int i=0; i<n; i++)
    for(int j=0; j<m; j++)
    {
        if(x[i]==y[j]) l[i+1][j+1]=l[i][j]+1;
        else l[i+1][j+1] = max(l[i][j+1],l[i+1][j]);
        res = max(res,l[i+1][j+1]);
    }
    cout<<res;
    return 0;
}
```

Do đặc trưng của c++, các kí tự trong xâu được đánh số từ 0 nên ta tính các chỉ số của mảng L[][] tịnh tiến lên 1 để không bị tràn mảng.

1.3.3. Tìm dãy con tăng dài nhất

Bài toán: Cho dãy số nguyên A = a_1, a_2, ..., a_n. ($n \leq 1000$, $-10000 \leq a_i \leq 10000$). Một dãy con của A là một cách chọn ra trong A một số phần tử giữ nguyên thứ tự. Như vậy A có 2^n dãy con.

Yêu cầu: Tìm dãy con đơn điệu tăng của A có độ dài lớn nhất.

Dữ liệu:

- Dòng đầu là số n
- Dòng thứ 2 chứa n số nguyên

Kết quả:

- Dòng đầu là độ dài dãy con tăng dài nhất
- Dòng thứ 2 là các số thuộc dãy con tăng dài nhất

Ví dụ: A = (1, 2, 3, 4, 9, 10, 5, 6, 7, 8). Dãy con đơn điệu tăng dài nhất là: (1, 2, 3, 4, 5, 6, 7, 8).

Giải:

Đặc trưng của bài toán là các cấu trúc con tối ưu, độ dài của dãy con tăng dài nhất ở vị trí i sẽ được tính qua độ dài dãy con tăng tại các vị trí j (j<i), gọi hàm mục tiêu f[i] là độ dài dãy con tăng dài nhất kết thúc ở vị trí i, ta sẽ xác định được:

- Mỗi dãy con có một phần tử sẽ là một dãy con tăng dài nhất có độ dài bằng 1. Vậy ban đầu f[i] = 1 với mọi i (bài toán cơ sở)
- Để tính độ dài dãy con tăng dài nhất kết thúc ở vị trí i, ta xem xét để ghép số ở vị trí i với các dãy con tăng dài nhất kết thúc ở vị trí j mà j<i; a[j]<a[i] và chọn ra dãy có độ dài lớn nhất. Ta được công thức truy hồi:

f[i] = max(f[j]) + 1; (j<i; a[j]<a[i])

- Bảng phương án là mảng f[]
- Độ dài dãy con tăng dài nhất sẽ là max của các f[]
- Truy vết: ta sử dụng một mảng t[i] = jmax (jmax là vị trí j mà tại đó f[j] đạt max theo công thức truy hồi), ta dùng một mảng res[] để lưu các số trong dãy con tăng dài nhất

Cài đặt:

daycontang.cpp

```cpp
#include <bits/stdc++.h>
#define maxn 1000
using namespace std;
int a[maxn+1],f[maxn+1],t[maxn+1], n, maxf=1,maxvt=1,res[maxn+1];
int main()
{
    cin>>n;
    for(int i=1; i<=n; i++){
        cin>>a[i];
        f[i]=1;
    }
    for(int i=2; i<=n; i++)
    for(int j=i-1; j>=0; j--)
    if(f[i]<f[j]+1&&a[j]<a[i]){
        f[i]=f[j]+1;
        t[i]=j;
        maxf=max(maxf,f[i]);
        maxvt=i;
    }
    cout<<maxf<<endl;
    int dem=0; int i=maxvt;
    while(i>0){
        dem++;
        res[dem]=a[i];
        i=t[i];
    }
    for(int i=dem; i>=1; i--) cout<<res[i]<<" ";
    return 0;
}
```

1.3.4. Bài toán cái túi

Bài toán: Có n đồ vật, vật thứ i có giá trị g_i và trọng lượng là w_i, cần xếp các đồ vật này vào cái túi có sức chứa trọng lượng tối đa là M sao cho tổng giá trị của túi là lớn nhất. Xác định giá trị lớn nhất đạt được.

Dữ liệu:

- Dòng đầu là hai số n, m
- N dòng tiếp theo mỗi dòng chứa 2 số lần lượt là giá trị và trọng lượng của vật thứ i

Kết quả: Đưa ra tổng giá trị lớn nhất đạt được

Đặc trưng của bài toán đó là: Ta thấy việc chọn hay không chọn đồ vật thứ i phụ thuộc vào cả 2 yếu tố đó là giá trị của đồ vật đó và trọng lượng của nó phụ thuộc vào trọng lượng tối đa tại thời điểm được chọn của túi

Ta xây dựng hàm F[i,j] là giá trị lớn nhất của túi khi xét từ mặt hàng thứ 1 đến mặt hàng thứ i và j là trọng lượng tối đa của túi, do vậy nghiệm tối ưu của bài toán chính là giá trị F[n,M].

Công thức truy hồi: Tính F[i,j] dựa trên nhận xét sau

F[0,j]=0 với j=0..M; (Bài toán cơ sở, nếu không chọn vật nào thì giá trị của túi bằng 0)

Nếu vật thứ i không được chọn thì F[i,j]= F[i-1,j] (Giá trị bằng giá trị của i-1 vật phía trước)

Nếu vật thứ i được chọn thì F[i,j]= G[i]+ F[i-1, j- W[i]] (chọn vật thứ i thì ta được giá trị của nó là G[i], và đồng thời khối lượng tối đa của túi sẽ còn lại là j- W[i])

Vì vậy việc chọn hay không chọn vật thứ i phụ thuộc vào giá trị cực đại của hai giá trị trên

F[i,j]= Max(F[i-1,j], G[i]+ F[i-1, j- W[i]]) (Công thức truy hồi)

Mảng F[][] là bảng phương án, kết quả của bài toán là F[n][M]

Cài đặt:

daycontang.cpp caitui.cpp

```cpp
#include <bits/stdc++.h>
#define maxn 1000
#define maxm 1000
using namespace std;
int g[maxn+1], w[maxn+1], f[maxn+1][maxm+1], n, m;
int main()
{
    cin>>n>>m;
    for(int i=1; i<=n; i++) cin>>g[i]>>w[i];
    for(int j=0; j<=m; j++) f[0][j]=0;
    for(int i=1; i<=n; i++)
    for(int j=1; j<=m; j++)
    {
        f[i][j]=f[i-1][j];
        if(j>w[i]) f[i][j] = max(f[i][j],g[i] + f[i-1][j-w[i]]);
    }
    cout<<f[n][m];
    return 0;
}

```

1.3.5. Bài toán di chuyển từ Tây sang Đông

Bài toán: Cho mảng 2 chiều a[n][m] trên đó ghi các số nguyên. Một người xuất phát tại một ô nào đó của cột 1 cần sang cột m theo nguyên tắc chỉ được di chuyển sang ô bên phải kề cạnh cùng hàng hoặc lệch nhau 1 hàng, sao cho tổng các số ghi trên đường đi là lớn nhất

Dữ liệu:

- Dòng đầu là hai số n, m
- N dòng tiếp theo mỗi dòng chứa m số nguyên

Kết quả: Ghi ra tổng giá trị lớn nhất đạt được

Đặc trưng của bài toán đó là: Ta phải tính được giá trị cực đại của các ô trên lưới dựa trên nhận xét sau:

Tại vị trí (i,j) của lưới giá trị cực đại nhận được chính bằng giá trị tại ô đó cộng với giá trị cực đại của các ô trước nó (i-1, j-1); (i, j-1);(i+1, j-1)

Công thức truy hồi: gọi B[i,j] là tổng cực đại các số trên đường đi từ cột 1 cho tới ô (i,j) khi đó B[1,j]=A[1,j] với j=1..m; và B[i,j]=Max(B[i-1,j-1], B[i,j-1], B[i+1,j-1])+ A[i,j]

Cài đặt:

```cpp
#include <bits/stdc++.h>
#define maxn 1000
using namespace std;
int a[maxn+1][maxn+1], b[maxn+1][maxn+1], n,m;
int main()
{
    cin>>n>>m;
    for(int i=1; i<=n; i++)
    for(int j=1; j<=m; j++)
    cin>>a[i][j];
    for(int i=1; i<=n; i++) b[i][1]=a[i][1];

    for(int j=2; j<=m; j++)
    for(int i=1; i<=n; i++)
    b[i][j] = max(b[i-1][j-1],max(b[i][j-1],b[i+1][j-1])) + a[i][j];

    int res = 0;
    for(int i=1; i<=n; i++) res = max(res, b[i][m]);

    cout<<res;
    return 0;
}
```

& others

Lưu ý là ta di chuyển lần lượt sang cột 2 rồi cột 3 rồi cứ thế đến cột m, do đó quá trình tính toán mảng B ta cần tính các giá trị lần lượt của các cột. Do vậy vòng for j qua các cột phải ở ngoài vòng for i chạy qua các dòng.

CHƯƠNG 2. MỘT SỐ BÀI TẬP ỨNG DỤNG QUY HOẠCH ĐỘNG GIẢI BẰNG PHƯƠNG PHÁP PHÂN TÍCH DỮ LIỆU CUỐI

2.1. Một số bài tập ứng dụng quy hoạch động

2.1.1. Phương pháp đơn vị dữ liệu cuối

Trong một lớp rất lớn các bài toán Tin học, chúng ta phải đếm các cấu hình tổ hợp có thứ tự (x_1, x_2,...,x_n) thỏa mãn một tính chất đặc trưng nào đó hoặc tìm giá trị tối ưu (theo một nghĩa nào đó) của các cấu hình này. Chúng ta sử dụng phương pháp quy hoạch động theo cách sau đây:

Tìm lời giải từng bước một: bước thứ k, đếm các cấu hình (x_1, x_2,...,x_k) hay tìm giá trị tối ưu của các loại cấu hình k phần tử (x_1, x_2,...,x_k).

Việc tìm lời giải từng bước một này thể hiện nguyên lý "chia để trị" trong tin học: Thay vì trực tiếp tìm lời giải bài toán lớn, ta giải quyết các bài toán nhỏ cùng mô hình, sau đó dựa trên kết quả các bài toán nhỏ để tìm kết quả bài toán lớn. Nói một cách nôm na, nó giống như việc chúng ta xây một bức tường thì phải xây từ móng, sau đó xây lần lượt các hàng gạch cho đến khi có một bức tường hoàn chỉnh, hàng gạch thứ k (tính từ móng trở lên) chính là hình ảnh trực quan của đơn vị dữ liệu cuối ở bước thứ k.

Đối với cấu hình ta xây dựng đến bước thứ k: (x_1, x_2,...,x_k) thì ta nói x_k là đơn vị dữ liệu cuối ở bước này. Việc xác định giá trị của x_k không hề khó khăn vì trước đó đã xây dựng được (x_1, x_2,...,x_{k-1}), từ đó có được phương án tính số lượng hay giá trị tối ưu cho bước thứ k.

Một điểm mấu chốt khi phân tích quy nạp xây dựng bước thứ k đó là: Nếu ta coi mỗi phần tử của cấu hình đang xây dựng như là một vị trí trên đường đi thì để đến được vị trí thứ k ta phải qua vị trí thứ k-1. Nói cách khác, để đến được vị trí cuối x_k đã xác định thì x_{k-1} phải là một trong các vị trí có thể "đi đến x_k". Từ nhận xét mấu chốt này mà việc phân tích cấu trúc x_k – đơn vị dữ liệu cuối bước k – để tìm ra những vị trí có thể của x_{k-1} chính là yếu tố quyết định để có được lời giải cho bài toán.

2.2.2. Cơ sở toán học

A/ Phương pháp quy nạp

"*Nếu một mệnh đề toán học P(n) phụ thuộc biến số tự nhiên n là đúng với giá trị cơ sở n = a, ngoài ra khi P(k) đúng kéo theo P(k+1) đúng thì mệnh đề P(n) đúng với mọi số tự nhiên n không nhỏ hơn a".*

Điều quan trọng nhất được áp dụng ở đây không phải dùng phương pháp quy nạp chứng minh bài toán mà chính là cách chứng minh P(k+1) đúng – để

làm điều này, trong phương pháp quy nạp người ta đã vận dụng thật sáng tạo cơ sở đã có đó là P(n) đúng với các n không quá k. Việc này hoàn toàn tương tự khi chúng ta phân tích xây dựng x_k theo các giá trị đã có trước đó trong cấu hình đang xây dựng của bài toán.

B/ Công thức truy hồi

Có hai dạng công thức truy hồi chúng ta sẽ sử dụng:

1/ Một dãy số hoàn toàn xác định khi biết k giá trị đầu tiên và công thức xác định một số hạng theo k số hạng liền trước nó:

{Fn} xác định nếu biết F_1, F_2,...,F_k và công thức $F_n = F(F_{n-1},F_{n-2},...,F_{n-k})$.

2/ Một dãy số hoàn toàn xác định khi biết giá trị đầu tiên và công thức xác định một số hạng theo các số hạng trước nó:

{Fn} xác định nếu biết F_1 và công thức $F_n = F(F_{n-1},F_{n-2},...,F_1)$.

Việc tính các giá trị ban đầu F_1, F_2, ..., F_k gọi là bước cơ sở, việc tìm công thức cho phép tính một số hạng của dãy theo các số hạng trước nó gọi là bước quy nạp.

2.2.3. Các bài toán ứng dụng phương pháp quy hoạch động

Sau đây là một số bài toán được chia theo các dạng khác nhau để thấy được những vận dụng phong phú của việc phân tích đơn vị dữ liệu cuối.

Dạng 1. Đếm các xâu, các cấu hình, các dãy số thõa mãn điều kiện nào đó

Bài toán 1: ĐẾM XÂU NHỊ PHÂN

Đề bài: Cho số nguyên dương n. Tính số xâu nhị phân độ dài n không có 2 chữ số 1 liền nhau.

Dữ liệu: vào từ file binary.inp gồm một số nguyên dương n

Kết quả: ghi ra file binary.out một số duy nhất là kết quả bài toán, vì kết quả có thể là một số rất lớn nên ta lấy kết quả là phần dư cho $10^9 + 7$

Ví dụ

Binary.inp	Binary.out
4	8

Xây dựng giải thuật:

Ta có thể liệt kê mọi xâu độ dài n không có 2 chữ số 1 liền nhau để đếm. Tuy nhiên phương án này rõ ràng không khả thi, sẽ không có đủ bộ nhớ và nhất là không có thời gian để làm công việc này khi n khá lớn vì lúc đó số lượng xâu cần liệt kê nhiều đến mức “kinh khủng”.

Từ nhận xét: Một xâu độ dài n thỏa mãn yêu cầu bài toán (không có 2 ký tự 1 liên tiếp) thì xâu con n-1 ký tự đầu tiên của nó cũng thỏa mãn điều này, thế có nghĩa là để có xâu độ dài n ta chỉ việc thêm 0 hoặc 1 vào sau xâu độ dài n-1 như vậy mỗi đơn vị dữ liệu là một ký tự và *đơn vị dữ liệu cuối* ở bước thứ k là ký tự thứ k của xâu. Từ đó bằng phương pháp phân tích quy nạp, ta có thể xây dựng công thức quy hoạch động cho phép tính số xâu độ dài n theo số xâu có độ dài nhỏ hơn cùng thỏa mãn yêu cầu bài toán.

Gọi F[k] là số xâu nhị phân độ dài k không có 2 chữ số 1 liền nhau.

Bước cơ sở: n = 1: có 2 xâu thoả mãn là '0' và '1' nên F[1]:=2.

n = 2: có 3 xâu thoả mãn là '00';'01'và '10' nên F[2]:=3.

Bước quy nạp: Giả sử đã tìm được các F[i] với i:=1,2…,k-1. Ta tính F[k] – số xâu độ dài k thỏa mãn bài toán có được do:

+ Thêm 0 vào sau 1 xâu độ dài k – 1, số xâu loại này bằng số xâu độ dài k-1 không có 2 chữ số 1 liền nhau và bằng F[k-1].

+ Thêm 1 vào sau 1 xâu độ dài k – 1, để tạo xâu độ dài k thoả mãn yêu cầu thì ký tự cuối cùng xâu độ dài k-1 phải là 0, do đó số xâu loại này bằng số xâu độ dài k-2 không có 2 chữ số 1 liền nhau và bằng F[k-2].

Vậy F[k] = F[k – 1] + F[k – 2] (1)

Do đã biết F[1], F[2] nên từ công thức này ta có thể tính F[n] với n tùy ý.

Độ phức tạp: do ta mất một vòng for để tính F[i] nên độ phức tạp là O(n).

Bài toán 2: ĐẾM XÂU NHỊ PHÂN MỞ RỘNG

Tương tự như trên, ta xây dựng công thức tính số xâu nhị phân độ dài n không có k chữ số 1 liền nhau (k>=2).

Ở đây ta cũng gọi F[n] để giữ giá trị số xâu nhị phân không có k chữ số 1 liền nhau và có độ dài n.

Bước cơ sở: Ta thấy F[0]:=1; F[i]:=2^i với i:=1,2,..k-1.

Với i:=k, F[k] = 2^k – 1 (trừ xâu có k chữ số 1)

Bước quy nạp:

+ Thêm 0 vào sau 1 xâu độ dài n – 1, số xâu loại này bằng số xâu độ dài n-1 không có 2 chữ số 1 liền nhau và bằng F[n-1].

+ Thêm 1 vào sau 1 xâu độ dài n – 1 cũng tạo thành F[n-1] xâu nhị phân mới có độ dài n nhưng trong đó có một số xâu không thoả mãn vì k chữ số liên tiếp cuối cùng bằng 1. Số xâu sau khi thêm 1 có k chữ số cuối bằng 1

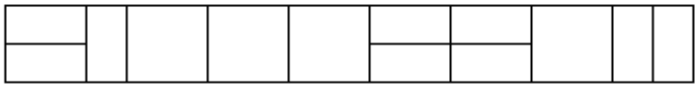
bằng số xâu độ dài n-1 thỏa mãn bài toán và có k-1 chữ số cuối cùng là 1 : $x_1x_2..x_{n-k-1}x_{n-k}11…1$, khi đó $x_{n-k} = 0$ và vì vậy số xâu loại này bằng số xâu $x_1x_2..x_{n-k-1}$ thỏa mãn bài toán và bằng F[n-k-1], từ đó số xâu độ dài n tận cùng bằng 1 thỏa mãn bài toán là: F[n-1] – F[n – k – 1] .

Vậy Ta có công thức quy hoạch động: F[n]:=2*F[n-1]-F[n-k-1].

Độ phức tạp: O(n) do có n vòng for.

Bài toán 3: LÁT VIỀN

Đề bài: Đường viền trang trí ở nền nhà có kích thước 2xN được lát bằng 2 loại gạch: loại kích thước 1x2 và loại 2x2. Hãy xác định số cách lát khác nhau có thể thực hiện.



Dữ liệu: Vào từ file văn bản TILE.INP, gồm nhiều dòng, mỗi dòng chứa một số nguyên N ($1 < N \leq 250$).

Kết quả: Đưa ra file văn bản TILE.OUT các kết quả tìm được module 1 000 000 007, mỗi số trên một dòng.

Ví dụ:

TILE.INP	TILE.OUT
2 8 12	3 171 2731

Xây dựng giải thuật:

Đương nhiên không thể liệt kê tất cả các cách lát rồi đếm bởi vì số cách lát quá lớn. Tuy nhiên khi để ý rằng mỗi cách lát, vị trí 1x2 cuối cùng của đường viền chỉ có thể bị phủ bởi một viên gạch 1x2 “đứng” , một viên gạch 2x2 hoặc 2 viên gạch 1x2 đặt “nằm”, ở đây “đơn vị dữ liệu cuối” của ta là cách đặt gạch lát vị trí cuối đường viền. Từ đó chúng ta lại nghĩ đến chuyện sẽ tính số cách lát đường viền độ dài n theo số cách lát đường viền ngắn hơn như sau:

Gọi F[k] là số lát viền kích thước 2*k.

Bước cơ sở: Ta thấy ngay F[1] = 1; F[2] = 3.

Bước quy nạp:

+ Nếu vị trí cuối cùng là 1 viên gạch 1x2 “đứng”: số cách lát bằng số cách lát viền kích thước 2*(k-1) và bằng F[k-1].

+ Nếu vị trí cuối cùng là 1 viên gạch 2*2 hay 2 viên gạch 1x2 xếp "nằm": số cách lát bằng số cách lát viền kích thước 2*(k-2) và bằng F[k-2].
Vậy F[k] = F[k-1] + 2*F[k-2].

Chú ý khi viết chương trình:

+ F[n] rất lớn nên phải thực hiện tính toán với số lớn.

+ Chỉ cần dùng ba biến tính luân phiên như **Bài toán 1**

Độ phức tạp: O(max(10^6, n))

Bài toán 4: ĐẾM SỐ DÃY CON TĂNG DÀI NHẤT

(Bài toán mở rộng của bài toán 1.3.3 – Tìm dãy con tăng đơn điệu dài nhất)

Bài toán: Cho dãy số nguyên A = a_1, a_2, ..., a_n. ($n \leq 1000$, $-10000 \leq a_i \leq 10000$). Một dãy con của A là một cách chọn ra trong A một số phần tử giữ nguyên thứ tự. Như vậy A có 2^n dãy con.

Yêu cầu: Tìm số lượng dãy con đơn điệu tăng của A có độ dài lớn nhất.

Dữ liệu:

- Dòng đầu là số n
- Dòng thứ 2 chứa n số nguyên

Kết quả:

- Dòng đầu là độ dài dãy con tăng dài nhất
- Dòng thứ 2 là số lượng dãy con tăng dài nhất

Ví dụ: A = (1, 2, 3, 4, 9, 10,11, 13, 5, 6, 7, 8). Dãy con đơn điệu tăng dài nhất là: (1, 2, 3, 4, 5, 6, 7, 8) hoặc (1, 2, 3, 4, 9, 10, 11, 13).

Ví dụ:

Cis.inp	Cis.out
12 1 2 3 4 9 10 11 13 5 6 7 8	8 2

Giải quyết bài toán mở rộng này có ý nghĩa sâu sắc trong việc rèn luyện tư duy quy hoạch động cho học sinh. Ở đây ta có nhận xét quan trọng rằng: a_{i1}, a_{i2}, ..., a_{ik} là dãy con tăng dài nhất thì mỗi đoạn đầu liên tiếp của nó: a_{i1}, a_{i2}, ..., a_{ip} (p<=k) sẽ là dãy con tăng dài nhất có số hạng cuối là a_{ip} như vậy có nghĩa p là độ dài dãy con tăng này nên nó phải bằng F[ip] (mảng F nói trên).

Từ đó ta sẽ tính số dãy con tăng T[i] có độ dài F[i] và số hạng cuối cùng là a_i. Rõ ràng a_i lúc này chỉ được thêm vào sau số hạng a_j bé hơn nó mà F[j] +1 =F[i]. Khi ấy tập các dãy con tăng độ dài F[i] kết thúc tại a[i] được bổ xung thêm T[j] phần tử. Vậy ở đây ta có công thức quy hoạch động như sau:

T[i] = Tổng các T[j] với j thỏa mãn: j < i; a[j] < a[i]; F[j] + 1 = F[i].

điều này có nghĩa là để tính T[i] ta phải tính trước các F[j], tất nhiên việc này sẽ kết hợp tính song song trong cùng một vòng lặp.

Kết quả cuối cùng là tổng các T[i] mà F[i] = max{F[j] : j = 1,2,...,n}.

Độ phức tạp: $O(n^2)$

Bài toán 5: CẶP SỐ 0

Xuất phát từ xâu ***S*** ban đầu chỉ chứa một ký tự '1', người ta biến đổi ***n*** lần theo quy tắc sau:

- Tạo xâu ***T*** bằng cách đảo các ký tự trong ***S***: '1' thành '0' và ngược lại,
- Tính ***S*** mới: ***S := T + S***.

Với cách biến đổi đó, ta có:

n	***S***
1	01
2	1001
3	01101001

Yêu cầu: Cho biết ***n*** ($0 < n \leq 10^5$). Hãy xác định cặp số 0 trong xâu S sau n lần biến đổi.

Dữ liệu: Vào từ file văn bản PAIR.INP gồm một số nguyên ***n***.

Kết quả: Đưa ra file văn bản PAIR.OUT kết quả lấy theo module $10^9 + 7$.

Ví dụ:

PAIR.INP
2
3

PAIR.OUT
1
1

Xây dựng giải thuật:

Do T là xâu bù của S và S:= T+S nên số lượng số cặp số 0 ở xâu mới được xác định như sau:

+ Số lượng cặp số 0 trong xâu S cũ;

+ Số lượng cặp số 0 trong xâu T = số lượng cặp số 1 trong xâu S cũ

+ x cặp số 0 tạo ra khi ghép 2 xâu S và T trong đó x=1 khi cuối xâu T và đầu xâu S cũ đều bằng 0, trường hợp ngược lại x=0.

Ta thấy rằng kí tự kết thúc xâu S luôn là kí tự ‘1’ nên kí tự kết thúc xâu T luôn là ‘0’ do đó để tính x ta chỉ cần cập nhập lại biến d lưu kí tự đầu của xâu là ‘0’ hay là ‘1’.

Gọi F[i] là số lượng cặp số 0, T[i] là số lượng cặp số 1 của S sau i lần biến đổi.

Bước cơ sở: F[1]= T[1] = T[2] = 0; F[2] = F[3]=T[3] = 1;

Bước quy nạp: F[i]:= T[i-1]+F[i-1]+x;

T[i]:=T[i-1]+F[i-1];

Độ phức tạp O(n).

Dạng 2. Tìm giá trị tối ưu của một dãy số thoả mãn điều kiện của bài toán

Bài toán 6: BỜM UỐNG RƯỢU

Đề bài: Bờm thắng phú ông trong một cuộc đánh cược và buộc phú ông phải đãi rượu. Phú ông bèn bày ra một dãy n chai chứa đầy rượu, và nói với Bờm rằng có thể uống bao nhiêu tuỳ ý, nhưng đã chọn chai nào thì phải uống hết và không được uống ở ba chai liền nhau bởi đó là điều xui xẻo.

Bạn hãy chỉ cho Bờm cách uống được nhiều rượu nhất.

Dữ liệu: Vào từ file văn bản BOTTLES.INP

Dòng đầu: ghi số nguyên dương n (n ≤ 100 000)

Các dòng tiếp ghi số nguyên dương (≤ 100 000) là dung tích của các chai rượu phú ông bày ra, theo thứ tự liệt kê từ chai thứ nhất đến chai thứ n, các số được ghi cách nhau bởi dấu cách hoặc dấu xuống dòng.

Kết quả: Ghi ra file văn bản BOTTLES.OUT

Ghi ra lượng rượu tối đa Bờm có thể uống.

Ví dụ

Bottles.inp	Bottles.out
6 16 10 10 13 10 30	69

Xây dựng giải thuật:

Mỗi đơn vị dữ liệu là một chai rượu, khi xét đến đơn vị dữ liệu cuối là chai rượu thứ K Bờm có thể uống hoặc không.

Gọi Drink[K] là lượng rượu tối đa uống được ở K chai đầu mà có uống chai thứ K;

NotDrink[K] là lượng rượu tối đa uống ở K chai đầu và không uống chai thứ K.

Khởi tạo NotDrink[1]:=0; Drink[1]:=A[1].

Khi đó với mỗi K từ 2 đến N:

Bờm không uống chai thứ K thì có thể uống hoặc không uống chai K-1:

NotDrink[K]:=max(NotDrink[K-1],Drink[K-1]);

Bờm uống chai thứ K thì không uống chai K-1 hoặc uống chai K-1 nhưng không uống chai K-2:

Drink[K]:=max{NotDrink[K-1];NotDrink[K-2] + A[K-1]} + A[K];

Độ phức tạp O(n).

Bài toán 7: TRÒ CHƠI VỚI BĂNG SỐ

Đề bài: Trò chơi với băng số là trò chơi tham gia trúng thưởng được mô tả như sau: Có một băng hình chữ nhật được chia ra làm *n* ô vuông, đánh số từ trái qua phải bắt đầu từ 1. Trên ô vuông thứ *i* người ta ghi một số nguyên dương a_i, $i = 1, 2, ..., n$. Ở một lượt chơi, người tham gia trò chơi được quyền lựa chọn một số lượng tùy ý các ô trên băng số. Giả sử theo thứ tự từ trái qua phải, người chơi lựa chọn các ô $i_1, i_2, ..., i_k$. Khi đó điểm số mà người chơi đạt được sẽ là:

$$a_{i1} - a_{i2} + ... + (-1)^{k-1}a_{ik}$$

Yêu cầu: Hãy tính số điểm lớn nhất có thể đạt được từ một lượt chơi.

Dữ liệu:

Dòng đầu tiên chứa số nguyên dương n ($n \leq 106$) là số lượng ô của băng số;

Dòng thứ hai chứa n số nguyên dương a1, a2, ..., an ($ai \leq 104$, $i = 1, 2, ..., n$) ghi trên băng số. Các số liên tiếp trên cùng dòng được ghi cách nhau bởi ít nhất một dấu cách.

Kết quả: Một số nguyên duy nhất là số điểm lớn nhất có thể đạt được từ một lượt chơi.

Ví dụ

Linegame.inp	Linegame.out
5 1 3 2 4 5	6

Xây dựng giải thuật:

Ta hình dung ra việc chọn lần lượt một dãy con để có được một tổng đan dấu lớn nhất. Mỗi đơn vị dữ liệu là một số hạng, số hạng a_k – đơn vị dữ liệu cuối bước k có thể chọn mang dấu + hoặc –.

Gọi FC[k] là tổng đan dấu lớn nhất mà số hạng cuối là a_k, FT[k] là tổng đan dấu lớn nhất mà số hạng cuối là - a_k . Ta có:

FC[k] = max{FT[i] + A[k], i < k};

FT[k] = max{FC[i] – A[k], i < k};

Cải tiến:

Việc tìm max{FT[i], i < k}; max{FC[i], i < k} được cập nhật ngay trong quá trình tính toán, vì vậy ta không cần hai mảng FT và FC mà chỉ cập nhật bằng hai biến F1, F2.

Độ phức tạp: O(n)

Bài toán 8: NỐI MẠNG MÁY TÍNH.

Các học sinh mỗi khi đến thực tập trong phòng máy tính thường hay chơi trò chơi trên mạng. Để ngăn ngừa, người trực phòng máy đã ngắt tất cả các máy ra khỏi mạng và xếp chúng thành một dãy trên một cái bàn dài và gắn chặt máy xuống mặt bàn rồi đánh số các máy từ 1 đến N theo chiều từ trái sang phải. Các học sinh tinh nghịch không chịu thua, họ đã quyết định tìm cách nối các máy trên bàn bởi các đoạn cáp nối sao cho mỗi máy được nối ít nhất với một máy khác. Để tiến hành công việc này, họ đã đo khoảng cách giữa hai máy liên tiếp. Bạn hãy giúp các học sinh này tìm cách nối mạng thỏa mãn yêu cầu đã nêu sao cho tổng độ dài cáp nối phải sử dụng là nhỏ nhất.

Dữ liệu: Vào từ file văn bản CABLE.INP:

Dòng đầu tiên chứa số lượng máy N ($1 \le N \le 25000$);

Dòng thứ i trong số N-1 dòng tiếp theo chứa khoảng cách từ máy i đến máy i+1 (i=1, 2, …, N-1). Biết rằng khoảng cách từ máy 1 đến máy N không vượt quá 10^6.

Kết quả: Ghi ra file văn bản CABLE.OUT độ dài của cáp nối cần sử dụng.

Ví dụ:

CABLE.INP	CABLE.OUT
6 2 2 3	7

2 2	

Xây dựng giải thuật:

Nhận thấy rằng: để tổng độ dài cáp nhỏ nhất thì ta không thể nối dây giữa các máy không liên tiếp; mỗi đơn vị dữ liệu là một máy, đơn vị dữ liệu cuối khi xét đến máy thứ k chính là máy này. Trong cách nối tối ưu máy thứ k có thể nối hoặc không nối với máy thứ k-1 mà vẫn thỏa mãn mỗi máy được nối với ít nhất một máy khác.

Gọi D[k] là khoảng cách giữa máy K-1 và máy K;

Link[K] là độ dài nhỏ nhất nối K máy đầu và K-1 nối với K;

NotLink[K] là độ dài nhỏ nhất nối K máy đầu và K-1 không nối với K.

Khởi tạo giá trị NotLink[1]:=0; Link[1]:=+∞ {+∞ ở đây có thể lấy là 10^7}.

Khi đó với mỗi K từ 2 đến N ta làm đồng thời hai công việc sau:

Nếu máy K được nối với máy K-1 thì máy K-1 nối hoặc không nối K-2 cũng được:

Link[K]:=min{Link[K-1];NotLink[K-1]} + D[K];

Nếu máy K không nối với máy K-1 thì máy K-1 phải nối với máy K-2:

NotLink[K]:=Link[K-1];

Do máy thứ nhất phải nối với máy thứ hai, máy thứ N-1 phải nối với máy thứ N kết quả là Link[N].

Cải tiến:

Ta thấy qua mỗi bước ta chỉ cần giữ lại 2 giá trị cuối của Link và NotLink nên ta có thể dùng 2 biến L và NL để thay thế cho 2 mảng. Như vậy ta còn có thể dùng biến D để thay cho mảng D. Khi đó ta chỉ dùng biến chạy K từ 2 đến N và biến tg vừa đọc dữ liệu vừa xử lý:

```
Readln(f,D);
tg:=NL;
NL:=L;
L:=min{tg;L} + D;
```

Độ phức tạp: O(n)

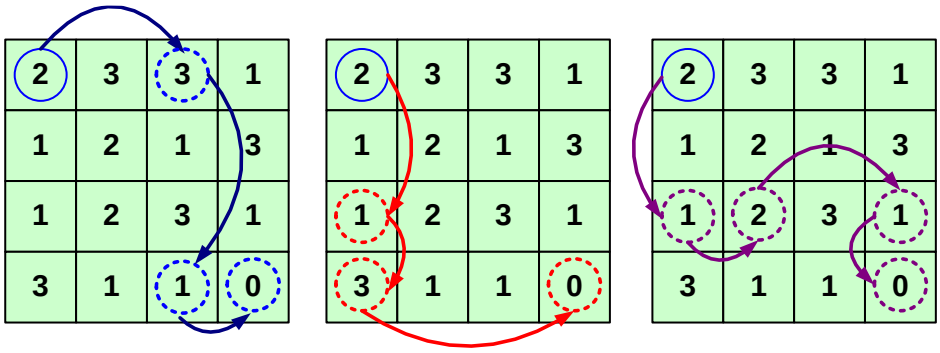
Dạng 3. Các bài toán di chuyển, cần tìm hành trình tối ưu theo điều kiện và yêu cầu của bài toán

Bài toán 9: NHẢY VỀ ĐÍCH (Bài toán mở rộng của bài toán 1.3.5)

Đề bài: Xét bảng kích thước ***n*×*n*** ô vuông ($4 \le n \le 100$), trên mỗi ô ghi một số nguyên trong phạm vi từ 0 đến 9. Ở góc trên trái có một quân cờ. Ta phải chuyển quân cờ về ô dưới phải của bảng theo các quy tắc sau:

- Chỉ được di chuyển trên một hàng sang phải hay xuống dưới theo một cột
- Số ghi trên ô có quân cờ kích thước bước nhảy
- Chỉ được di chuyển trong phạm vi bảng đang xét.

Với một bảng cho trước, có thể không có cách nhảy từ ô trên trái về ô dưới phải hoặc có nhiều cách. Ví dụ, với bảng cho ở hình dưới, ta có 3 cách khác nhau nhảy về đích.



Yêu cầu: Cho ***n*** và các số trên bảng. Hãy xác định số cách nhảy khác nhau về đích.

Dữ liệu: Vào từ file văn bản JUMP.INP:Dòng đầu tiên chứa số nguyên ***n***, ***n*** dòng sau; mỗi dòng chứa n số nguyên mô tả một dòng của bảng theo thứ tự từ trên xuống dưới.

Kết quả: Đưa ra file văn bản JUMP.OUT một số nguyên duy nhất – số cách nhảy. Kết quả có thể là số rất lớn nên ta lấy phần dư cho $10^9 + 7$.

Ví dụ

JUMP.INP	JUMP.OUT
4 2 3 3 1 1 2 1 3 1 2 3 1 3 1 1 0	3

Xây dựng giải thuật:

Suy nghĩ đầu tiên là đường đi từ ô (1;1) đến ô (N;N) cũng là một dãy các ô tương tự đường đi trong bài toán 1.3.5, có điều quy luật di chuyển của quân cờ này là kiểu khác. Mặc dù vậy ta hoàn toàn có thể sử dụng ý tưởng: Ta sẽ tính số cách đi của quân cờ từ ô (1;1) đến mọi ô khác lần lượt trên các dòng từ 1 đến N từ trái sang phải. Khi đó đơn vị dữ liệu cuối chính là ô cuối cùng của mỗi đường đi.

Gọi F[i,j] là số cách nhảy khác nhau từ ô [1,1] đến ô [i,j] theo quy tắc. Kết quả cuối cùng là F[N,N].

Các F[i,j] được tính lần lượt từ trên xuống dưới từ trái sang phải. Nhận xét tương tự rằng để đến được ô [i,j] thì trước đó quân cờ phải đứng ở một trong các ô [p,q] thỏa mãn:

p = i (cùng hàng) và q + a[p,q] = j;

hoặc q = j (cùng cột) và p + a[p,q] = i;

Khi đó F[i,j] là tổng các F[p,q] thỏa mãn một trong hai điều kiện trên:

Bước cơ sở: F[1,1] = 1; F[i,j] = 0 khi i<=0 hoặc j<= 0;

Bước quy nạp:

```
Theo cột:    for p:=i-1 downto i-9 do   {do a[p,q] không vượt quá 9}
              if a[p,j]=(i-p) then      {nhảy được từ ô[p,j] đến ô [i,j]}
                F[i,j]:=F[i,j]+F[p,j];
Theo hàng:   for q:=j-1 downto j-9 do
             if a[i,q]=(j-q) then       {nhảy được từ ô [i,q] đến ô [i,j] }
             F[i,j]:=F[i,j]+F[i,k];
```

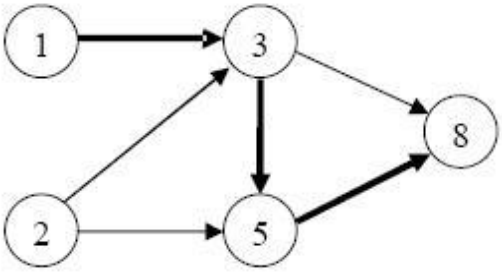
Độ phức tạp $O(n^3)$

Bài toán 10: LÒ CÒ

Nhảy lò cò là trò chơi dân gian của Việt Nam. Người trên hành tinh X cũng rất thích trò chơi này và họ đã cải biên trò chơi này như sau: Trên mặt phẳng vẽ n vòng tròn được đánh số từ 1 đến n. Tại vòng tròn i người ta điền số nguyên dương a_i. Hai số trên hai vòng tròn tùy ý không nhất thiết phải khác nhau. Tiếp đến người ta vẽ các mũi tên, mỗi mũi tên hướng từ một vòng tròn đến một vòng tròn khác. Quy tắc vẽ mũi tên là: Nếu có ba số a_i, a_j, a_k thỏa mãn $a_k = a_i + a_j$ thì vẽ mũi tên hướng từ vòng tròn i đến vòng tròn k và mũi tên hướng từ vòng tròn j đến vòng tròn k. Người chơi chỉ được di chuyển từ một vòng tròn đến một vòng tròn khác nếu có mũi tên xuất phát từ một trong số các vòng tròn, di chyển theo cách mũi tên đã vẽ để đi đến các vòng tròn

khác. Người thắng cuộc sẽ là người tìm được cách di chuyển qua nhiều vòng tròn nhất.

Ví dụ: Với 5 vòng tròn và các số trong vòng tròn là 1, 2, 8, 3, 5, trò chơi được trình bày trong hình dưới đây:



Khi đó có thể di chuyển được nhiều nhất qua 4 vòng tròn (tương ứng với đường di chuyển được tô đậm trên hình vẽ).

Yêu cầu: Hãy xác định xem trong trò chơi mô tả ở trên, nhiều nhất có thể di chuyển được qua bao nhiêu vòng tròn.

Dữ liệu:

Dòng đầu chứa số nguyên n ($3 \le n \le 1000$);

Dòng thứ hai chứa dãy số nguyên dương a1, a2, ..., an ($ai \le 109$, i=1, 2,..., n).

Hai số liên tiếp trên một dòng được ghi cách nhau bởi dấu cách.

Kết quả: Ghi ra số lượng vòng tròn trên đường di chuyển tìm được.

Xây dựng giải thuật:

Giả sử các số trong vòng tròn được cho trong mảng a[1..N]. Từ một vòng tròn sẽ chỉ có thể đi đến một vòng tròn ghi số lớn hơn, vì vậy ta sẽ sắp xếp lại mảng a (tương ứng với các vòng tròn) theo chiều tăng dần. Bây giờ ta có quy tắc di chuyển: từ vòng tròn i đến được vòng tròn j nếu a[j] = a[i] + a[k] với k <> i, k < j nào đó. Với quy tắc di chuyển này, mỗi đơn vị dữ liệu là một vòng tròn (ứng với một số trong dãy đã sắp xếp). Khi xét đơn vị dữ liệu cuối là vòng tròn thứ i, để nó là vị trí cuối của một dãy nhiều vòng tròn nhất ta thêm nó vào sau dãy dài nhất có thể các vòng tròn phía trước.

Gọi F[i] là số lượng vòng tròn nhiều nhất có thể đi qua tới vòng tròn thứ i (tính cả vòng tròn thứ i).

Bước cơ sở: F[i] = 1 với mọi i= 1,2,…,N

Bước quy nạp: Để tính F[i] ta thêm vòng tròn i vào sau dãy nhiều vòng tròn nhất đã đi qua ở phía trước, có nghĩa là tìm F[j] lớn nhất thỏa mãn điều

kiện có đường đi từ vòng tròn j đến vòng tròn i, lúc đó F[i] = F[j] + 1. Như đã nêu, việc kiểm tra xem có đường đi từ vòng tròn j đến vòng tròn i chính là việc liểm tra xem có thể tách a[i] thành tổng a[j]+a[k], điều này có thể thực hiện nhanh chóng bằng tìm kiếm nhị phân giá trị a[i] – a[j] trong dãy a[1..j-1]

Độ phức tạp: $O(n^2*logn)$.

2.2. Xây dựng chương trình và bộ test để kiểm tra tính tối ưu giải thuật của một số bài toán

2.2.1. Bài Toán 1: ĐẾM XÂU NHỊ PHÂN

Cài đặt

```
#include <bits/stdc++.h>
#define mod 1000000007
using namespace std;
long long f[100005]; int n;
int main()
{
   //sinh();
   freopen("binary.inp","r",stdin);
   freopen("binary.out","w",stdout);
   cin>>n;
   f[1]=2;
   f[2]=3;
   for(int i=3; i<=n; i++) f[i] = (f[i-1]+f[i-2])%mod;
   cout<<f[n];
   return 0;
}
```

Hàm sinh test:

```
void sinh(){
   srand(time(0));
   freopen("binary.inp","w",stdout);
   n =rand()%100000+2;
   cout<<n;
}
```

2.2.2. Bài Toán 2: ĐẾM XÂU NHỊ PHÂN MỞ RỘNG

Cài đặt chương trình

```
#include <bits/stdc++.h>
#define mod 1000000007
using namespace std;
```

```
long long f[100005]; int n,k;
int main()
{
   //sinh();
   freopen("binaryex.inp","r",stdin);
   freopen("binaryex.out","w",stdout);
   cin>>n>>k;
   f[0]=1;
   for(int i=1; i<k; i++) f[i]=f[i-1]*2;
   f[k]=f[k-1]*2 -1;
   for(int i=k+1; i<=n; i++) f[i] = (2*f[i-1]%mod-f[i-k-1])%mod;
   cout<<f[n];
   return 0;
}
```

Hàm sinh test

```
void sinh(){
   srand(time(0));
   freopen("binaryex.inp","w",stdout);
   n =rand()%100000+2;
   k =rand()%100000+2;
   if(n<k) swap(n,k);
   cout<<n<<" "<<k;
}
```

2.2.3. Bài Toán 3: LÁT VIỀN

Cài đặt chương trình

```
#include <bits/stdc++.h>
#define mod 1000000007
using namespace std;
long long f[1000005]; int n;
int main()
{
   //sinh();
   freopen("tile.inp","r",stdin);
   freopen("tile.out","w",stdout);
   f[1]=1;
   f[2]=3;
   for(int i=3; i<=1e6; i++) f[i] = (f[i-1]+2*f[i-2]%mod)%mod;
```

```
    while(cin>>n){
        cout<<f[n]<<endl;
    }
    return 0;
}
```

Hàm sinh test case

```
void sinh(){
    srand(time(0));
    freopen("tile.inp","w",stdout);
    n = 1000;
    while(n--) cout<<rand()%1000000+1<<endl;
}
```

2.2.4. Bài toán 4: ĐẾM SỐ DÃY CON TĂNG DÀI NHẤT

Cài đặt chương trình

```
#include <bits/stdc++.h>
#define maxn 1000
using namespace std;
int a[maxn+5], f[maxn+5], t[maxn+5], n, res=0,c=0;
int main()
{
    //sinh();
    freopen("cis.inp","r",stdin);    freopen("cis.out","w",stdout);
    cin>>n;
    for(int i=1; i<=n; i++){
        cin>>a[i];      f[i]=1;       t[i]=1;
    }
    for(int i=1; i<=n; i++){
        for(int j=i-1; j>=1; j--){
            if(a[j]<a[i]){
                if(f[j]+1>f[i]){f[i]=f[j]+1;t[i]=t[j];}
                else if(f[j]+1==f[i]) t[i]+=t[j];
            }
        }
        res=max(res,f[i]);
    }
    for(int i=1; i<=n; i++) if(f[i]==res) c+=t[i];
    cout<<res<<endl<<c;
```

```
    return 0;
}
```

Hàm sinh test case

```
void sinh(){
    srand(time(0));
    freopen("cis.inp","w",stdout);
    n = 1000;
    cout<<n<<endl;
    for(int i=1; i<=n; i++) cout<<rand()%100000+1<<" ";
}
```

2.2.5. Bài toán 5: CẶP SỐ 0

Cài đặt chương trình

```
#include <bits/stdc++.h>
#define MOD 1000000007
using namespace std;
long long a[1000005],f[1000005], t[1000005];int n,d,x;
int main(){
   sinh();
   freopen("pair.inp","r",stdin);
   freopen("pair.out","w",stdout);
   f[1]=t[1]=t[2]=0;
   f[2]=f[3]=t[3]=1;
   cin >>n;
   for (int i=4;i<=n;i++){
      d=(i%2!=0);
      x=(d+1)%2;
      f[i] = (f[i-1]+t[i-1]+x)%MOD;
      t[i] =(t[i-1]+f[i-1])%MOD;
   }
   cout<<f[n];
   return 0;
}
```

Hàm sinh test

```
void sinh(){
   freopen("pair.inp","w",stdout);    srand(time(0));
   n=100000;
   cout<<n<<endl;
```

```
}
```

2.2.6. Bài toán 6: BỜM UỐNG RƯỢU

Cài đặt chương trình

```
#include <bits/stdc++.h>
#define maxn 100000
using namespace std;
long long a[maxn+5],n,res=0;
long long f[maxn+5]; //Drink
long long g[maxn+5]; //NotDrink
int main()
{
   //sinh();
   freopen("bottles.inp","r",stdin);
   freopen("bottles.out","w",stdout);
   cin>>n;
   for(int i=1; i<=n; i++) cin>>a[i];
   f[1]=a[1]; g[1]=0;
   for(int i=2; i<=n; i++) {
      g[i] = max(f[i-1],g[i-1]);
      f[i] = max(g[i-1],a[i-1]+g[i-2])+a[i];
   }
   res=max(f[n],g[n]);
   cout<<res<<endl;
   return 0;
}
```

Hàm sinh test

```
void sinh(){
   srand(time(0));
   freopen("bottles.inp","w",stdout);
   n = 100000;
   cout<<n<<endl;
   for(int i=1; i<=n; i++) cout<<rand()%100000+1<<" ";
}
```

2.2.7. Bài toán 7: TRÒ CHƠI VỚI BĂNG SỐ

Cài đặt chương trình

```
#include <bits/stdc++.h>
```

```
using namespace std;
long long n,f1,f2,a;
int main()
{
   sinh();
   freopen("linegame.inp","r",stdin);
   freopen("linegame.out","w",stdout);
   cin>>n;
   cin>>f1;
   for(int i=2; i<=n; i++){
      cin>>a;
      long long tg=f1;
      f1=max(f1,f2+a);
      f2=max(f2,tg-a);
   }
   cout<<max(f1,f2);
   return 0;
}
```

Hàm sinh test

```
void sinh(){
   srand(time(0));    freopen("linegame.inp","w",stdout);
   n = 10;
   cout<<n<<endl;
   for(int i=1; i<=n; i++) cout<<rand()%100000+1<<" ";
}
```

2.2.8. Bài toán 8: NỐI MẠNG MÁY TÍNH.

Cài đặt chương trình

```
#include <bits/stdc++.h>
using namespace std;
long long n, NL, L,d;
int main()
{
   sinh();
   freopen("cable.inp","r",stdin);
   freopen("cable.out","w",stdout);
   cin>>n;
   NL=0; L=1e9;
```

```
    for(int i=2; i<=n; i++){
        cin>>d;
        long long tg=NL;
        NL = L;
        L = min(tg,L) + d;
    }
    cout<<L;
    return 0;
}
```

Hàm sinh test

```
void sinh(){
    srand(time(0));    freopen("cable.inp","w",stdout);
    n = 100000;    cout<<n<<endl;
    for(int i=2; i<=n; i++) cout<<rand()%100000+1<<" ";
}
```

2.2.9. Bài toán 9: NHẢY VỀ ĐÍCH (Bài toán mở rộng của bài toán 1.3.5)

Cài đặt chương trình

```
#include <bits/stdc++.h>
#define MOD 1000000007
using namespace std;
int a[1005][1005],f[1005][1005];int n;
int main(){
    //sinh();
    freopen("jump.inp","r",stdin) ;    freopen("jump.out","w",stdout) ;
    cin >>n;
        for (int i=1;i<=n;i++)for (int j=1;j<=n;j++){cin >> a[i][j];f[1][1]=1;}
        for(int i=1;i<=n;i++) for(int j=1;j<=n;j++)
        {
            for(int p=1;p<i;p++)if(a[p][j]==i-p)f[i][j]=(f[i][j]+f[p][j])%MOD;
            for(int q=1;q<j;q++)if(a[i][q]==j-q)f[i][j]=(f[i][j]+f[i][q])%MOD;
        }
    cout<<f[n][n];

}
```

Hàm sinh test

```
void sinh(){
    freopen("jump.inp","w",stdout);    srand(time(0));
```

```
    n=100;cout<<n<<endl;
    for(int i=1; i<=n; i++) {
        for(int j=1; j<=n; j++) cout<<rand()%3+1<<" ";    cout<<endl;
    }
}
```

2.2.10. Bài toán 10: LÒ CÒ

Cài đặt chương trình

```
#include <bits/stdc++.h>
#define MOD 1000000007
using namespace std;
int a[2005],f[2005];int n,res=0;
int main(){
    sinh();
    freopen("hop.inp","r",stdin);
    freopen("hop.out","w",stdout);
    cin >>n;
    for(int i=1; i<=n; i++){
        cin>>a[i]; f[i]=1;
    }
    sort(a+1,a+n+1);
    for (int i=1;i<=n;i++){
        for(int j=i-1; j>=1; j--){
            int vt = lower_bound(a+1,a+i,a[i]-a[j])-a;
            if(vt!=j&&a[vt]==a[i]-a[j]) f[i]=max(f[i],f[j]+1);
        }
    }
    for(int i=1; i<=n; i++) res=max(res,f[i]);

    cout<<res;
    return 0;
}
```

Hàm sinh test

```
void sinh(){
    srand(time(0));    freopen("hop.inp","w",stdout);
    n = 100000;    cout<<n<<endl;
    for(int i=1; i<=n; i++) cout<<rand()%100000+1<<" ";
}
```

KẾT LUẬN VÀ KIẾN NGHỊ

1. Kết luận

Đề tài đã nghiên cứu được một số bài toán điển hình sử dụng phương pháp quy hoạch động để giải, thấy được tính ưu việt hơn so với các phương pháp khác ở một số bài tập cụ thể. Đề tài đã tìm kiếm, lựa chọn và mở rộng được một số bài tập khác có sử dụng phương pháp quy hoạch động để giải, xây dựng code, đồng thời viết chương trình sinh bộ test tự động để kiểm tra tính chính xác và tối ưu của giải thuật, sử dụng phần mềm Themis để chấm điểm và kiểm tra kết quả của chương trình.

Đề tài đã góp phần nâng cao năng lực sư phạm của tác giả, và định hướng triển khai áp dụng vào giảng dạy học sinh giỏi Tin học dự thi cấp tỉnh cho học sinh trường PTTHSP Tràng An.

Trong thời gian tới, nhóm tác giả sẽ tiếp tục nghiên cứu sâu tìm kiếm nhiều bài toán giải bằng phương pháp quy hoạch động và tổng quát hóa một số dạng toán hay gặp có thể giải bằng phương pháp này. Song song với đó, nhóm tác giả sẽ tiếp tục tìm hiểu thêm các phương pháp và chuyên đề tin học khác để đáp ứng được yêu cầu của phạm vi chương trình thi học sinh giỏi Tin học phổ thông trong tỉnh Ninh Bình.

2. Kiến nghị

Phương pháp quy hoạch động rất hiệu quả cho lớp bài toán tối ưu, tuy nhiên không phải bài toán tối ưu nào cũng có thể giải quyết bằng phương pháp này. để giải quyết bài toán cần linh hoạt dựa trên việc tính toán so sánh độ phức tạp của giải thuật và chương trình cần xử lý được các giá trị test lớn từ đó giáo viên lựa chọn và sử dụng các phương pháp như quy hoạch động, tham lam, quay lui, nhánh cận hay đệ quy, … một cách hợp lý.

TÀI LIỆU THAM KHẢO

[1] Hồ Sĩ Đàm, *Tài liệu giáo khoa chuyên tin quyển 1*, NXB Giáo dục Việt Nam, 2009.

[2] Lê Minh Hoàng, Ebook *Giải thuật và lập trình,* Đại học sư phạm Hà Nội, 2002.

[3]. Trần Ngọc Anh https://topdev.vn/blog/thuat-toan-quy-hoach-dong/, xem 20/10/2022