НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ

Факультет компьютерных наук, Образовательная программа «Прикладная математика и информатика» Программный проект

«Подготовка задач для олимпиад школьников»

Выполнил Деб Натх Максим, БПМИ181

Руководитель: Густокашин Михаил Сергеевич, директор Центра студенческих олимпиад, ФКН ВШЭ

Описание предметной области

Предметная область проекта — разработка задач для соревнований по программированию, в том числе состязаний для школьников по программированию, проводимых НИУ ВШЭ, а именно, «Высшая проба по информатике», «Московская олимпиада школьников по информатике» .

Основные понятия, определения, термины

- *Задача по олимпиадному программированию* — проблема, для решения которой необходимо придумать и реализовать алгоритм. Задача считается решённой, если участник смогли составить программу, правильно работающую на тестовых примерах, подготовленных жюри.

Основные понятия, определения, термины

- *Задача по олимпиадному программированию* — проблема, для решения которой необходимо придумать и реализовать алгоритм. Задача считается решённой, если участник смогли составить программу, правильно работающую на тестовых примерах, подготовленных жюри.
- *Проверяющая программа (checker)* — программа, сверяющая вывод на конкретном тестовом примере ответ участника и ответ жюри и возвращающая вердикт по данному тестовому примеру.

Основные понятия, определения, термины

- *Задача по олимпиадному программированию* — проблема, для решения которой необходимо придумать и реализовать алгоритм. Задача считается решённой, если участник смогли составить программу, правильно работающую на тестовых примерах, подготовленных жюри.
- *Проверяющая программа (checker)* — программа, сверяющая вывод на конкретном тестовом примере ответ участника и ответ жюри и возвращающая вердикт по данному тестовому примеру.
- *Валидатор (validator)* — программа, проверяющая корректность входных данных.

Основные понятия, определения, термины

- *Задача по олимпиадному программированию* — проблема, для решения которой необходимо придумать и реализовать алгоритм. Задача считается решённой, если участник смогли составить программу, правильно работающую на тестовых примерах, подготовленных жюри.
- *Проверяющая программа (checker)* — программа, сверяющая вывод на конкретном тестовом примере ответ участника и ответ жюри и возвращающая вердикт по данному тестовому примеру.
- *Валидатор (validator)* — программа, проверяющая корректность входных данных.
- *Генератор (generator)* — программа, генерирующя тестовые примеры.

Актуальность работы

Ежегодно в России проходит несколько десятков перечневых олимпиад по программированию, для их проведения которых необходим набор оригинальных задач. Задачи эти должны затрагивать различные области математики, компьютерных наук, алгоритмов и структур данных.

«Высшая проба по информатике» и *«Московская олимпиада школьников по программированию»* входят в число таких олимпиад. Они проводятся при поддержке центра студенческих олимпиад ФКН.

Цель и задачи работы

Целью проекта была подготовка пакета нескольких задач по спортивному программированию, которые будет возможно использовать вместе с системами *ejudge* или *Яндекс.Контест*.

Задачи:

1. Разработка задачи, её анализ, разработка её решения

Цель и задачи работы

Целью проекта была подготовка пакета нескольких задач по спортивному программированию, которые будет возможно использовать вместе с системами *ejudge* или *Яндекс.Контест*.

Задачи:

1. Разработка задачи, её анализ, разработка её решения
2. Реализация решения задачи на одном из поддерживаемых языков программирования

Цель и задачи работы

Целью проекта была подготовка пакета нескольких задач по спортивному программированию, которые будет возможно использовать вместе с системами *ejudge* или *Яндекс.Контест*.

Задачи:

1. Разработка задачи, её анализ, разработка её решения
2. Реализация решения задачи на одном из поддерживаемых языков программирования
3. Реализация вспомогательных программ — валидатора, чекера, генераторов

Цель и задачи работы

Целью проекта была подготовка пакета нескольких задач по спортивному программированию, которые будет возможно использовать вместе с системами *ejudge* или *Яндекс.Контест*.

Задачи:

1. Разработка задачи, её анализ, разработка её решения
2. Реализация решения задачи на одном из поддерживаемых языков программирования
3. Реализация вспомогательных программ — валидатора, чекера, генераторов
4. Создание группы тестов для проверки решений

Анализ существующих решений

В силу технологических особенностей олимпиада «Высшая проба» проводится на платформе *ejudge*[1], «Московская олимпиада школьников» проводится на платформе *Yandex.Contest*[2].

В связи с этим задачи необходимо будет готовить в формате пакетов, поддерживаемых этими системами.

«Codeforces Polygon» [3] — единственная крупная и наиболее популярная система подготовки задач, её формат является де-факто стандартом в подготовке задач. Поддерживает экспортирование на платформы Yandex.Contest и ejudge.

---

[1] https://ejudge.ru

[2] https://contest.yandex.ru

[3] https://polygon.codeforces.com

Функциональные и нефункциональные требования

1. Работоспособность при использовании тестирующей системой подготовленного пакета задач.
2. Возможность отправить произвольное решение в систему и получить вердикт по нему (`OK`, `WA`, `PE`, `TL`, `ML`, `CE`).
3. Корректная работа тестирующей программы на правильных и неправильных решениях.

Описанные выше требования продиктованы функциональными особенностями систем и общепринятыми стандартами.

4. Требование в решении задачи знаний в использовании алгоритмов и структур данных, умения пользоваться языками программирования.
5. Нетривиальность решения.

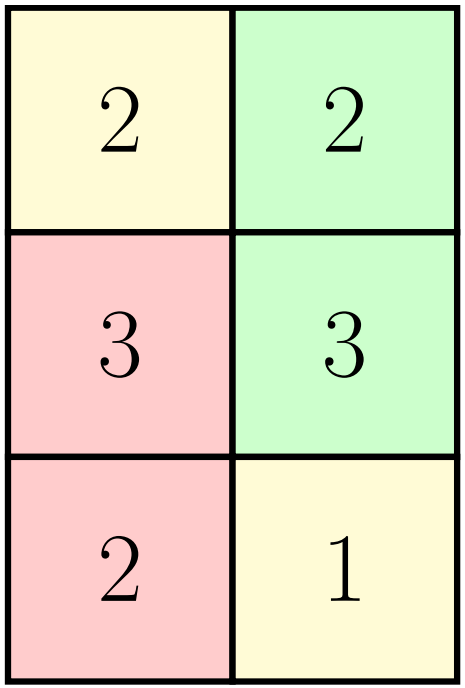
Задача №1

Задача была предложена на втором отборочном этапе олимпиады «Высшая проба». Этот этап проходил в качестве онлайн-тура длительностью 3 часа на платформе ejudge.

Задача №1

Условие

Рис.: $N = 3, K = 2$



- Дана полоска (матрица) натуральных чисел размера $2 \times N$ и натуральное число K.
- Требуется разместить на данной полоске ровно K непересекающихся костей домино (матриц 2×1 или 1×2) таким образом, чтобы сумма чисел на непокрытых костями клетках была минимальна.
- Требуется найти восстановить любую конфигурацию, на которой достигается этот минимум.

Задача №1

Условие

Ограничения тестов:

- $1 \leqslant N \leqslant 2 \cdot 10^5$;
- $0 \leqslant K \leqslant 2 \cdot 10^5$;
- $0 \leqslant N \times K \leqslant 2 \cdot 10^5$;
- $K \leqslant N$;

Ограничения на решение задачи:

- Ограничение по виртуальной памяти: 256MB
- Ограничение по виртуальному времени: 1 сек.

Задача №1

Решение

- Заметим, что задача минимизации суммы непокрытых клеток равносильна задаче максимизации суммы покрытых клеток.
- Воспользуемся методом многомерного динамического программирования.
- Обозначим за `dp[k][i][j]`, где $0 \leq i \leq N$, $0 \leq j \leq K$, $0 \leq k \leq 3$ максимальную сумму покрытых клеток, если рассматривается задача покрытия первых i столбцов j костями, при этом в последнем ряду:

$$\begin{array}{ll} \text{ни одна клетка не покрыта} & \text{, если } k = 0; \\ \text{только первая клетка покрыта} & \text{, если } k = 1; \\ \text{только вторая клетка покрыта} & \text{, если } k = 2; \\ \text{обе клетки покрыты} & \text{, если } k = 3; \end{array}$$

- В таком случае максимальная сумма будет равна $\max\limits_{0 \leq k \leq 3}$ `dp[k][N][K]`.

Задача №1

Решение

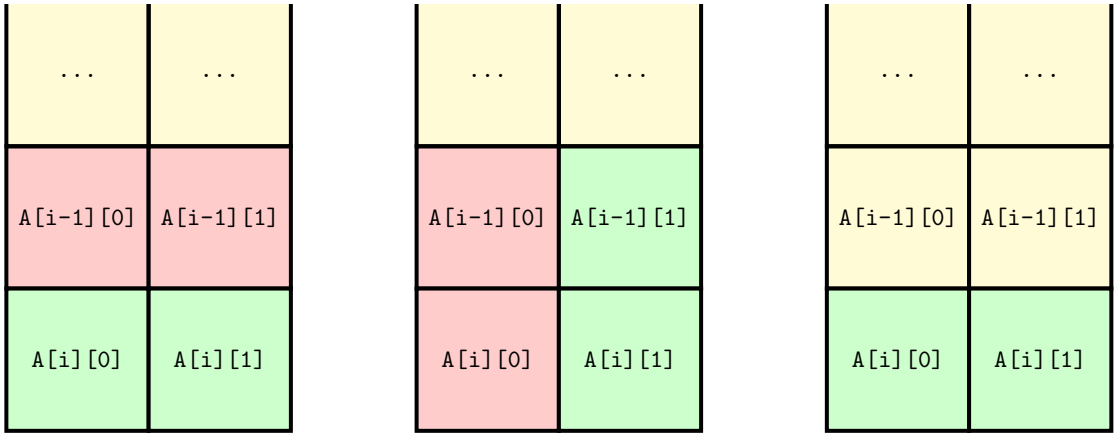
- По определению `dp` имеем, что

$$\texttt{dp[0][i][j]} = \max_k \texttt{dp[k][i-1][j]}$$

Задача №1

Решение

- Есть два взаимозаменяемых способа заполнить последний ряд костями (b и c):



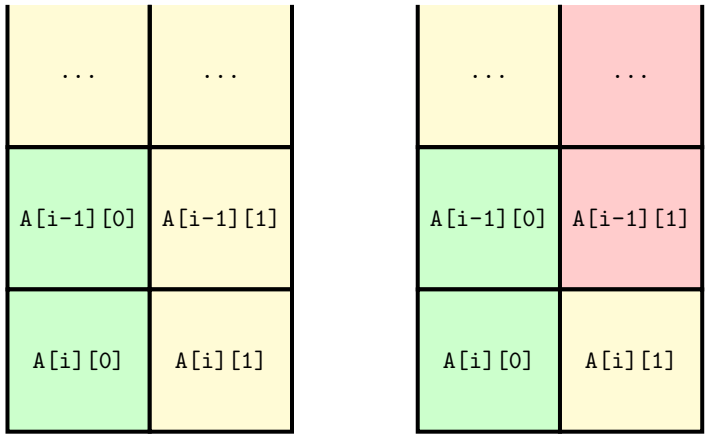
- Если в решении используется конфигурация b, её можно заменить на a. Значит, можно считать, что мы всегда в 3 случае последний ряд заполняем одной костью, откуда имеем:

$$\texttt{dp[3][i][j]} = \max_{0 \leqslant k \leqslant 3} \texttt{dp[k][i-1][j-1]} + \sum_{0 \leqslant k \leqslant 1} \texttt{A[i][k]}$$

Задача №1

Решение

- Если рассматривается случай, в котором только первая из двух клеток последнего ряда занята (случай 2), то есть два случая: когда клетка `[i-1][1]` покрыта и не покрыта костью:



- Отсюда имеем:

$$dp[1][i][j] = \max(dp[0][i-1][j-1],\ dp[2][i-1][j-1]) + \\ + A[i][0] + A[i-1][0]$$

- Аналогично выводится формула для `dp[2][i][j]`.

Задача №1

Решение

- Положим изначально
  - `dp[k][i][j]` $= -\infty$;
  - `dp[0][0][0] = 0;`
  - `dp[3][0][1] = A[0][0] + A[0][1];`
- С помощью вышеуказанных формул, можем тогда за $\mathcal{O}(NK)$ найти ответ.

Задача №1

Реализация

- Чекер к данной задаче тривиален — считывая и проверяя на валидность решения жюри и участника, программа сверяет найденную сумму и возвращает вердикт — `OK`, если значение целевой функции совпало, `WA`, если жюри нашло ответ лучше и `FAIL` иначе.
- Авторское решение реализует вышеописанную логику, работает за $\mathcal{O}(NK)$, написано на C++, что при ограничениях $N \times K \leqslant 2 \cdot 10^5$ работает на максимальном тесте не более чем за 0.2 сек.

Кусок кода с описанной выше логикой:

```
for (int i = 0; i < 4; i++) {
    dp[i].resize(
        n,
        vector<int>(k + 1, -INF)
    );
}

dp[0][0][0] = 0;
dp[3][0][1] = A[0][0] + A[0][1];

for (int i = 1; i < n; i++) {
    dp[0][i][0] = 0;
    for (int j = 1; j <= k; j++) {
        dp[0][i][j] = max(
            dp[0][i-1][j],
            dp[1][i-1][j],
            dp[2][i-1][j],
            dp[3][i-1][j]
        );

        dp[1][i][j] = max(
            dp[0][i-1][j-1],
            dp[2][i-1][j-1]
        ) + A[i][0] + A[i-1][0];

        dp[2][i][j] = max(
            dp[0][i-1][j-1],
            dp[1][i-1][j-1]
        ) + A[i][1] + A[i-1][1];

        dp[3][i][j] = max(
            dp[0][i-1][j-1],
            dp[1][i-1][j-1],
            dp[2][i-1][j-1],
            dp[3][i-1][j-1]
        ) + A[i][0] + A[i][1];
    }
}
int ans = max(
    dp[0][n-1][k],
    dp[1][n-1][k],
    dp[2][n-1][k],
    dp[3][n-1][k]
);
```

Задачи №2-3

- Задачи 2 и 3 имели *открытые тесты* — они известны участнику и надо найти ответы к ним. Задачи носили оптимизационными задачами — точный ответ неизвестен и работа оценивается в сравнении с лучшим известным решением.
- Задача были дана на отборочном и финальном этапах олимпиады «Московская олимпиада школьников»[4]. Раунды проходил на платформе Яндекс.Контест.
- Длительность отборочного этапа — несколько месяцев, финального этапа — 4 часа.

[4] http://mos-inf.olimpiada.ru/

Задача №2

Условие

- Вариация известной NP-полной задачи SETCOV.
- Дано множество A из n элементов (все элементы – числа от 1 до n).
- Дано семейство B из m подмножеств A. i-е подмножество содержит ровно k_i элементов, равных $x_{i\,1}, x_{i\,2}, \ldots x_{i\,k_i}$.
- Требуется выбрать какое-то подсемейство попарно непересекающихся множеств с максимальной мощностью объединения.

Задача №2

Решения

- Задачу можно детерминировано решить с помощью полного перебора за $\mathcal{O}\left(n \cdot 2^m\right)$.
- Если использовать `std::bitset` для хранения подмножеств, можно улучшить асимптотику до $\mathcal{O}\left(\frac{n2^m}{\omega}\right)$, где ω — длина машинного слова.
- Если m слишком велико, можно использовать рандомизированный алгоритм: выберем из данных m множеств какое-то подсемейство размера $m_0 < m$ и решим задачу для этого подсемейства за $\mathcal{O}\left(\frac{n2^{m_0}}{\omega}\right)$. Повторим операцию несколько раз и найдём среди найденных ответов лучший.

Задача №2

Решения

- Задачу можно решать как задачу оптимизации с помощью известных методов решения задачи оптимизации.
- Простейший вариант: поддерживать найденный ответ (подсемейство) и среди невключённых в него подмножеств пытаться включить какое-то множество, например, случайное или то, которое даёт наибольший прирост в значении целевой функции.
- Можно использовать *метод имитации отжига* (simulated annealing), где модификации состояния — это включения и исключения каких-то подсемейств из ответа.

Задача №2

Решения

- Построим ориентированный граф, где вершины — это всевозможные подмножества множества A. Тогда проведём ребро из вершины $U \subset A$ в вершину $V \subset A$ тогда и только тогда, когда существует множество $C \subset B$, такое, что $U \sqcup C = V$.
- Путь из вершины $\varnothing$ в вершину V говорит о том, что множество V можно набрать подмножествами из B.
- Построим граф и найдём максимальную достижимую вершину с помощью алгоритма обхода в глубину за $\mathcal{O}\left(\frac{nm \cdot 2^n}{\omega}\right)$

Задача №2

Реализация

- Было сгенерировано 20 тестов со следующими значениями *n* и *m*:

№	n	m	№	n	m
1	5	5	11	20	100
2	10	10	12	24	100
3	20	15	13	10000	10
4	100	20	14	10000	11
5	100	20	15	1000	30
6	100	20	16	1000	60
7	20	20	17	1000	60
8	20	40	18	9999	100
9	20	60	19	10000	100
10	20	100	20	10000	100

Задача №2

Реализация

- В качестве оценки решения участника была выбрана формула

$$5 \times \left(\frac{\text{ParticipantSolution}}{\text{BestSolution}} \right)^3$$

  где ParticipantSolution — значение целевой функции в решении участника, а BestSolution — лучший найденный ответ.
- Чекер к данной задаче тривиален — считывая и проверяя на валидность решения жюри и участника, программа сверяет найденные ответы и возвращает балл по формуле выше.
- Главное авторское решение реализует алгоритмы, описанные в разделе с решениями.

Задача №3

Условие

- Сокобан — логическая игра-головоломка, в которой игрок передвигает ящики по лабиринту, показанному в виде плана, с целью поставить все ящики на заданные конечные позиции.
- Сокобан может двигаться вверх, вниз, влево и вправо. Он не может проходить сквозь стены или ящики. Он может толкать только одну коробку за раз.
- Известна конфигурация лабиринта — поля $n \times m$, состоящая из пустых клеток или стен. Также известна начальная позиция каждого из ящиков, конечные позиции, куда надо поставить ящики и начальное положение Сокобана.
- Необходимо найти кратчайший способ решить головоломку — найти наиболее короткий (по числу действий) способ поставить все ящики на позиции.

Задача №3

Условие

Пример головоломки и её решения:

Задача №3

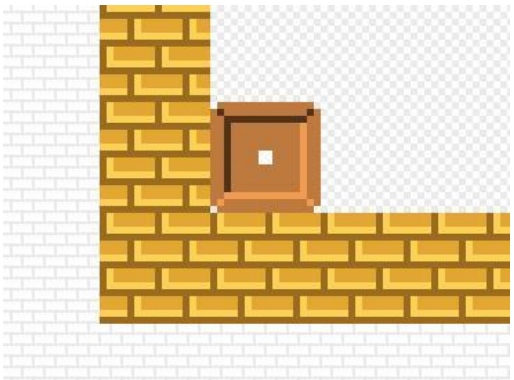
Решение

- Рассмотрим граф, состоящий из всевозможных конфигураций (конфигурация – это совокупность положений ящика и сокобана), где ребро ставится из вершины A в вершину B тогда и только тогда, когда из конфигурации A при шаге сокобана в одном из направлений, лабиринт перейдёт в состояние B.
- Обозначим за S стартовое состояние, а за $\{T_i\}$ — все конечные состояния. Проведём рёбра из $\{T_i\}$ в фиктивную вершину T.
- Задача свелась к задаче поиска кратчайшего пути из S в T.

Задача №3

Решение

- В качестве базового решения можно рассмотреть *BFS* (поиск в ширину). Его решение оправдано, так как хотя граф и может быть очень большим, ответ практически всегда достаточно мал.
- Можно оптимизировать *BFS* таким образом, чтобы не рассматривать гарантированно тупиковые конфигурации, например, те, в которых ящик стоит в углу — оттуда достать его будет невозможно:



Задача №3

Решение

- Можно оптимизировать BFS, с помощью метода *meet-in-the-middle*:
  - Запустим два алгоритма BFS: в начальном графе из стартовой вершины и в транспонированном графе из конечной вершины
  - Будем поочерёдно запускать по фазе BFS (*d*-я фаза находит все вершины, расстояние до которых равно *d*) в одном алгоритме и другом.
  - Если они оба найдут кратчайший путь до какой-то вершины *v*, то будет найден кратчайший путь до вершины *v* из *S* и кратчайший путь до *T* из вершины *v*
  - Можно показать, что объединение этих путей будет искомым кратчайшим путём.

Задача №3

Реализация

- Было решено, что решения, использующие простой BFS будут набирать 30% баллов, а двунаправленный BFS — все баллы.
- Для генерации тестов были выбраны уже известные головоломки сокобан, порядка 3 тысяч
- Авторское решение было протестировано на каждом из них, замеряя время работы обычного BFS, двухстороннего BFS, а также объём потребляемой памяти.
- В первую группу (15 тестов) были отобраны тесты, суммарное время работы на которых обычного BFS не превосходило 2 минут. Во вторую группу (25 тестов) были отобраны тесты, суммарное время работы полного решения на которых не превосходило 8 минут, и при этом решение с использование обычного BFS не работало в связи с ограничениями по памяти или времени.

Задача №3

Реализация

- В качестве оценки решения участника была выбрана формула

$$2 \times \left(\frac{\text{BestSolution}}{\text{ParticipantSolution}} \right)^4$$

  где ParticipantSolution — длина пути в решении участника, а BestSolution — лучший найденный путь.
- Чекер к данной задаче тривиален — считывая и проверяя на валидность решения жюри и участника (эмулируя работу головоломки), программа сверяет найденные ответы и возвращает балл по формуле выше.
- Главное авторское решение реализует алгоритм, описанные в разделе с решениями.

Задача №3

Реализация

- Главное решение не хранит граф явно, а генерирует все рёбра только тогда, когда обрабатывает очередную вершину
- Вершины хранятся как массивы чисел, где первое число — положение сокобана, а остальные числа — сортированные положения коробок.
- Положение — число от 0 до 255 (предполагается, что все поля достаточно маленькие). Для хранения массивов положений используется `std::pair<long long, long long>` (в качестве массива из 16 байт)
- Это сделано для оптимизации времени и памяти, используемых программой.

Тестирование проекта

Перед проведением олимпиад проходит *«прорешивание»* задач заинтересованными людьми — организатором олимпиады, членами методической комиссии, студентами. Их отзывы по подготовленным задачам были приняты во внимание, недочёты исправлены.

Основные результаты работы

1. Были подготовлены 3 задачи, используя которые были проведены туры перечневых олимпиад по программированию.
2. Первая задача была предложена более чем 1000 участникам, вторая — более чем 2500, третья — более чем 300.
3. Подготовленные пакеты функционировали как и предполагалось, значимых неполадок в течении тура не наблюдалось.
4. По 3 задаче 45 участников отправляли решения, по 2 задаче было предпринято свыше 1900 посылок.
5. Большинство задач оказались не слишком простыми и не слишком сложными — в их решении требовались знания, связанные со спортивным программированием, равно как и умение оптимально писать код.

Направления дальнейшей работы

Разработанные задачи нельзя назвать тупиковыми: так, каждую из них можно упростить или усложнить и переиспользовать в дальнейшем.

1. Например, если рассмотреть 1 задачу но уже на полоске $N \times M$ вместо $N \times 2$, концептуально решение не поменяется, но аналогичное решение, работая за $\mathcal{O}(KN2^M)$, будет гораздо сложнее в реализации.
2. Во 2 и 3 задачах можно уменьшать и увеличивать ограничения на входные данные, требуя тем самым более сложные или оптимальные решения.

Спасибо за внимание.

Деб Натх Максим

mdebnatkh@edu.hse.ru
debnatkh@gmail.com

Москва, 2020